中国研究生创新实践系列大赛
“华为杯”第十七届中国研究生数学建模竞赛

学　　校	同济大学
参赛队号	20102470319
队员姓名	1.李英超
	2.赵业成
	3.许艳红

中国研究生创新实践系列大赛
“华为杯”第十七届中国研究生
数学建模竞赛

题 目　面向康复工程的脑电信号分析和判别模型

摘　　要：

大脑是人体中高级神经活动的中枢，拥有着数以亿计的神经元，并通过相互连接来传递和处理人体信息，借助于技术手段对信号进行处理和分析有助于生理健康的判断和神经性疾病的监测，比如通过机器学习和数据挖掘，对大量数据进行训练学习从而找出规律进行分类识别，对数据进行反复的训练和测试有利于进一步提高识别精准率，为临床治疗提供重要的辅助作用。

借助于数据预处理，支持向量机、随机森林、卷积神经网络等机器学习方法就目标识别、通道选择、样本训练和测试、睡眠分期预测问题进行了数学建模和编程尝试，以求更好的结合实际解决问题。

问题一先将目标识别问题转化为判别每一周期是否出现 P300 信号的问题，将原始数据滤波处理成训练数据 X 和测试数据 C，并将训练数据 X 进行随机排列、倍增等处理，再将训练数据 X 进一步区分为训练集 A 和测试集 B。接着比较三种机器学习算法（支持向量机、随机森林、卷积神经网络）在训练集 A 上训练并在测试集 B 上测试效果，选择用卷积神经网络对测试数据 C 进行预测。同时发现对 S1 受试者的目标识别准确率始终低于受试者 S2-S5，在不考虑受试者 S1 识别效果的前提下，统计 S2 到 S5 的 10 个待识别目标预测情况，最终 10 个待识别目标的预测结果为 MF52ITKXAO。

问题二主要使用了卷积神经网络算法进行单个被试者通道选择设计，利用卷积核权重来衡量 20 个通道的重要程度，针对单个个体进行特定分析，将权重值较高的通道筛选出来，结果发现通道选择情况因人而异，每个受试者的较优通道组合有一定的差异，同时又有一定的相似性，在训练过程中我们发现 S4 和 S5 受试者脑电信号特征较为明显，受噪声干扰小，所以将 S4 和 S5 较优通道全部纳入选择，另外如果一个通道被 3 个以上的受试者选中，那么该通道也将纳入选择，由此选出一组最优通道组合（一共 15 个通道）：1,2,3,4,5,6, 7,8,9,10,11,13,14,16,19。为了进一步判断通道选择质量好坏，我们只考虑该组最优通道，再次对 S1-S5 受试者训练数据集合中的测试样本进行目标识别分析，发现相较于原始 20 个通道全部考虑的情况，通道选择后的识别准确率有一定程度的提高，从而验证了该最优通道组合比较适合所有受试者。

问题三主要采用了标签传播算法和自适应半监督学习算法进行监督学习和半监督学习的比较，标签传播算法的训练结果不收敛，故舍弃该方法。自适应半监督学习算法训练结果显示，采用自适应半监督学习算法的预测准确率有一定提升，预测稳定性有大幅提升。通过对 S1 到 S5 的前五个字母预测准确程度的辨别，我们选择了对前五个字母预测较为准确的预测信息进行对后五个字母的预测，结果为 TKXAO，且预测的一致性比较高。

问题四主要使用了神经网络和支持向量机两种方法对睡眠分期进行预测，基于训练样本比例和预测准确率考虑，将训练样本数要求较少预测准确率较高的支持向量机方法用来

设计预测模型，随机的从每个睡眠分期中选择等量样本进行训练，通过观察预测精准率随样本量比例的变化情况，发现训练集使用比例在 0.1 到 0.2 和 0.8 到 0.9 两区间内上升趋势明显，而比例在 0.2-0.8 区间范围内趋势平缓（测试集准确率随训练样本增加不明显），由此我们认为使用训练集比例为 0.2 时，可保证在使用较少训练集数据的前提下获得良好的分类效果，即最终用支持向量机方法以占总体样本 **0.2 比例**的训练样本实现了 71%左右的预测准确率。

关键字：滤波处理 卷积神经网络 主成分分析 支持向量机 半监督学习 随机森林

目录

一、问题重述

1.1 问题背景

脑电信号（Electroencephalograph,EEG）是人脑的生理电信号,如图 1.1.1 所示，这些信号包含了大量的生理信息，是研究学者通过对脑电信号处理进行神经系统疾病和症状判断的主要依据，尤其在癫痫病的诊断和治疗过程中发挥了重要作用。通过人-机互动的方式来获取脑电信号，之后再对数据进行进一步的分析和处理。一般采集脑电信号的主要工具为脑-计算机接口(Brain-Computer Interface,BCI),简称脑机接口,是指不依赖于脑的正常输出通路(即外周神经和肌肉)的脑-机(计算机或其他装置)通讯系统。

本题中所使用的事件相关电位 P300 是应用最广泛的内源性事件相关电位，是在事件(如听觉、视觉刺激)发生后大约 300ms 出现的一个正向波，最经典的应用是 Farewell 和 Don-chin 在 1988 年提出并设计的字符拼写器简称为 P300 Speller，如图 1.1.2 所示，使用 26 个英文字母和 0-9 个数字以及下划线排列成的 6×6 的字符矩阵。实验过程中提示被试者注视“目标字符”，之后进行字符矩阵的闪烁模式，每次以随机的顺序闪烁字符矩阵的一行或一列，闪烁时长为 80 毫秒，间隔为 80 毫秒；最后，当所有行和列均闪烁一次后，则结束一轮实验。在被试者注视“目标字符”的过程中，当目标字符所在行或列闪烁时，脑电信号中会出现 P300 电位；而当其他行和列闪烁时，则不会出现 P300 电位。根据是否会出现 P300 电位来判断该行或列是否出现目标字符，从而进一步判断找出待识别目标。

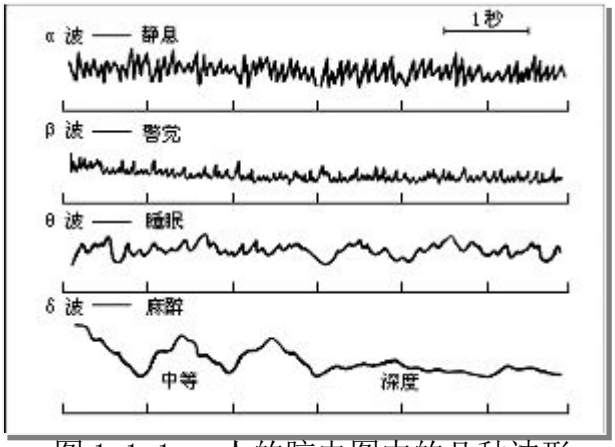


图 1.1.1： 人的脑电图中的几种波形

图 1.1.2：P300 Speller 矩阵

在脑电信号分析的过程中，主要是对信号进行特征提取和分类研究，用到的方法多种多样，较为常见的有频域分析、时域分析、小波变换、人工神经网络分析、非线性动力学分析，也有一些论文研究基于独立分量分析和支持向量机相结合对脑电信号进行处理[1]。

1.2 问题分析

脑电信号分析一共有四个问题需要进行研究，借助于数据挖掘知识对数据进行分析和处理，寻找信号背后的规律，找出合适的机器学习方法对待识别目标进行精准识别，进一步判断实验过程中的冗余信号通道，训练脑电信号数据找出识别模型，同时借助于睡眠数据对不同睡眠状态期进行识别分类。

问题一：在脑-机接口系统中既要考虑目标的分类准确率，同时又要保证一定的信息传输速率。根据 5 个被试者的训练和测试数据，设计或采用一个方法，在尽可能使用较少轮次（要求轮次数小于等于 5）的测试数据的情况下，找出 5 个被试者测试集中的 10 个待

识别目标，并给出具体的分类识别过程，可与几种方法进行对比，来说明设计方法的合理性。

问题二：由于采集的原始脑电数据量较大，这样的信号势必包含较多的冗余信息。在20个脑电信号采集通道中，无关或冗余的通道数据不仅会增加系统的复杂度，且影响分类识别的准确率和性能。请分析所给5个被试者数据，并设计一个通道选择算法，给出针对每个被试的、更有利于分类的通道名称组合（要求通道组合的数量小于20大于等于10，每个被试所选的通道可以不相同）。基于通道选择的结果，进一步分析对所有被试者都较适用的一组最优通道名称组合，并给出具体分析过程。为了方便参赛者对最优通道组合进行选择，赛题给出了测试数（char13-char17）的结果，它们的字符分别是：M、F、5、2、1。

问题三：在P300脑-机接口系统中，往往需要花费很长时间获取有标签样本来训练模型。为了减少训练时间，请根据5个被试者的训练和测试数据，选择适量的样本作为有标签样本，其余训练样本作为无标签样本，在问题二所得一组最优通道组合的基础上，设计一种学习的方法，并利用问题二的测试数据（char13-char17）检验方法的有效性，同时利用所设计的学习方法找出测试集中的其余待识别目标（char18-char22）。

问题四：根据睡眠脑电数据.xlsx中所给出的特征样本，请设计一个睡眠分期预测模型，在尽可能少的训练样本的基础上，得到相对较高的预测准确率，给出训练数据和测试数据的选取方式和分配比例，说明具体的分类识别过程，并结合分类性能指标对预测的效果进行分析。

二、模型假设与符号说明

2.1 模型基本假设

首先对数据集使用进行一定的说明：由于所给数据中 S2_test_event.xlsx 中缺少char22的event信息，S3_test_data.xlsx缺少char22的原始数据，因此问题一中S1、S4、S5被试者数据集中待识别目标为10个（char13-char22）,S2、S3被试者数据集中待识别目标为9个（char13-char21），同时问题三中 S1、S4、S5 被试者数据集中待识别目标为10个（char13-char22）,S2、S3被试者数据集中待识别目标为9个（char13-char21）。

为了建模过程的顺利开展，构建模型时考虑了以下假设条件：

假设一：本论文中出现的“train”是指机器学习各类算法中所需要的训练集（样本），以下统称为训练集 A，“test”是指机器学习各类算法中所需要的测试集（样本），以下统称为测试集B。而非问题所给数据附件1中的“train.xlsx”和“test.xlsx”，为避免混淆，附件1中数据集称为“训练数据X”，以下统称为和“测试数据C”。

假设二：在无噪声干扰下，人类脑电信号特征是类似的，这样可以由单个样本的特征中总结出群体规律特征，便于信号通道的筛选。

2.2 模型符号说明

表 2.2.1 模型符号说明

符号	符号说明
S1-S5	5 位被测试者编号
train_data	训练用数据
train_event	训练数据事件标签
test_data	测试用数据
test_event	测试数据事件标签
char01-char12	已知目标字符
char13-char22	待识别目标字符

三、数据预处理

人体是一个复杂的有机体，生理过程受许多非人为控制因素的影响，脑电信号是通过电极在头皮表面采集到的反映大脑内部状态的生物电信号，但是在 EEG 采集过程中，不可避免地会引入一些并非来自脑神经细胞的各种伪迹和干扰，如眼电、肌电、心电、工频干扰等，特别是眼电干扰，幅度较大，眼电信号波形如下图所示。而待分析的特定脑电信号则通常幅值较小，所以在其他信号的干扰下脑电信号的非平稳性特点比较突出，因此在对目标进行分类识别之前需要进行一定的预处理。研究过程中将 20 个信号通道所收集的数据进行去趋势和滤波处理，以求信号输出较为平稳，便于进一步分析。

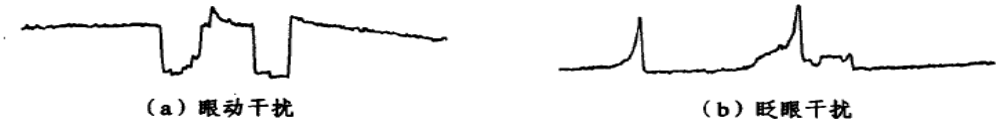
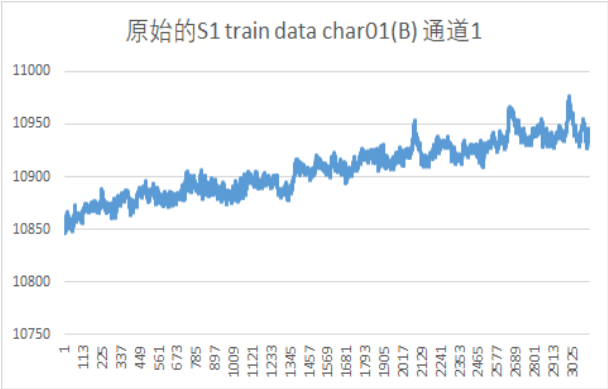


图 3.0 眼电信号波形

3.1 去趋势

考虑到脑-机接口在获取数据时可能会造成数据偏移的情况，从数据中删除趋势可以更好的将分析集中在数据趋势本身的波动上，数据去趋势就是对数据减去一条最优（最小二乘）的拟合直线、平面或曲面，使去趋势后的数据均值为零。因此我们在进行问题分析之前借助于 Matlab 软件对数据进行去趋势处理，运用 Detrend 函数分别进行了一次函数拟合和二次函数拟合来分析效果，分别就通道 1 和通道 11 采集数据进行去趋势前后结果对比，通道 1 情形如图 3.1.1 和图 3.1.2 所示，通道 11 情形如图 3.1.3 和图 3.1.4 所示，我们发现与初始数据相比，去趋势得到的信号图波动程度仍较大，因此去趋势效果作用不大，所以在数据处理的过程中省略去趋势这一步骤。



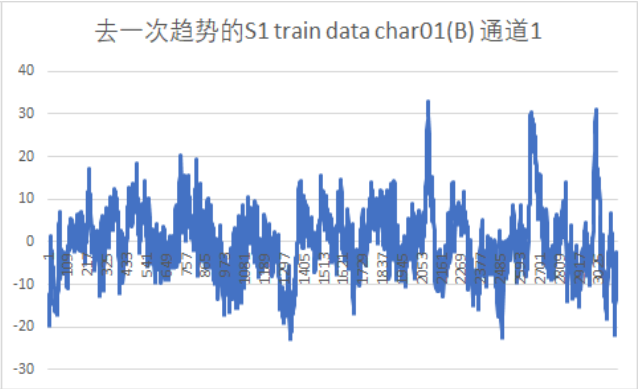
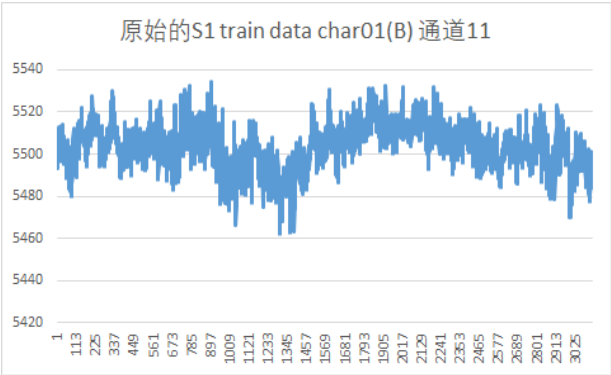


图 3.1.1 通道 1 原始脑电信号数据波动情况　图 3.1.2 通道 1 去一次趋势后数据波动情况



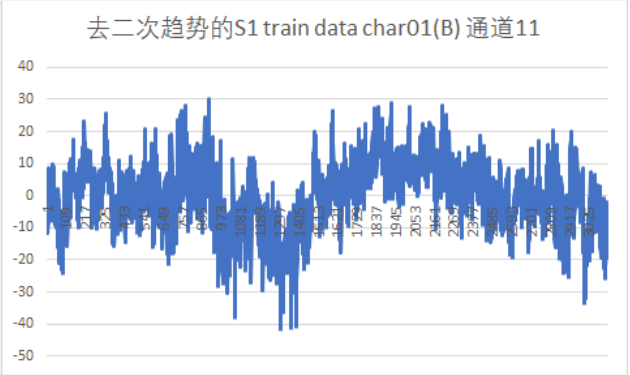


图 3.1.3 通道 1 原始脑电信号数据波动情况　图 3.1.4 通道 1 去一次趋势后数据波动情况

3.2 滤波

由于在脑电信号的采集过程中，会受到周围噪声的干扰以及工频噪声的影响[2]，所以为了得到平稳的脑电信号输出，需要对原始信号进行噪声的滤除，利用带通滤波器，将 1Hz - 20Hz 频率之间的信号取出。滤波处理主要是借助于 eeglab 软件，主要操作过程如下所示：

在 eeglab 界面上，选择 Tools > Filter the data > Basic FIR filter，输入 1Hz 作为下边缘频率,20Hz 作为上边缘频率，然后点击"OK".参数设计过程如下图 3.2.1 所示：

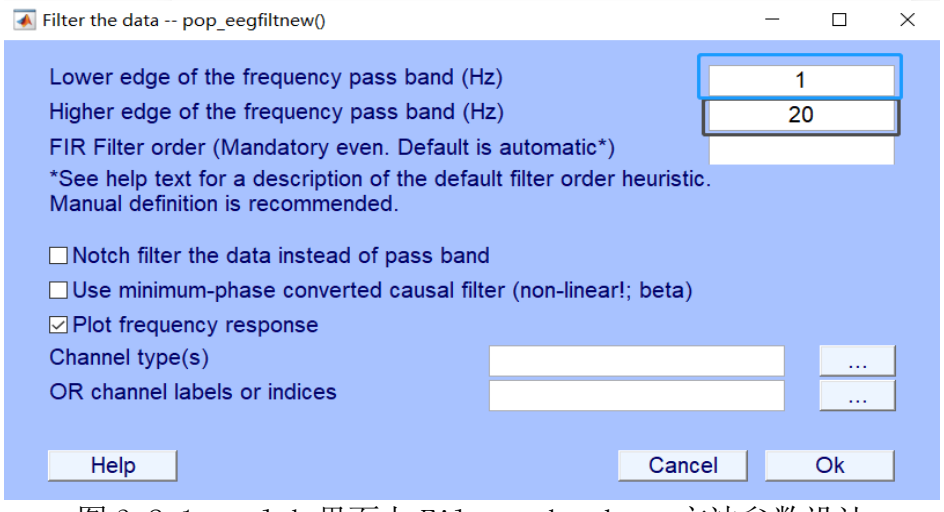


图 3.2.1 eeglab 界面上 Filter the data 方法参数设计

进行完上述的操作后，S1_train_data 中 char06（Q）的 20 个通道的采集信号滤波前

后对比图如下所示，滤波前由于不同通道间取值范围相差较大，一张图表中无法显示所有通道信号数据，因此无法判断信号的平稳性等特征，滤波后所有通道信号数据展现的更加清晰和明显，有利于进行进一步特征判断。

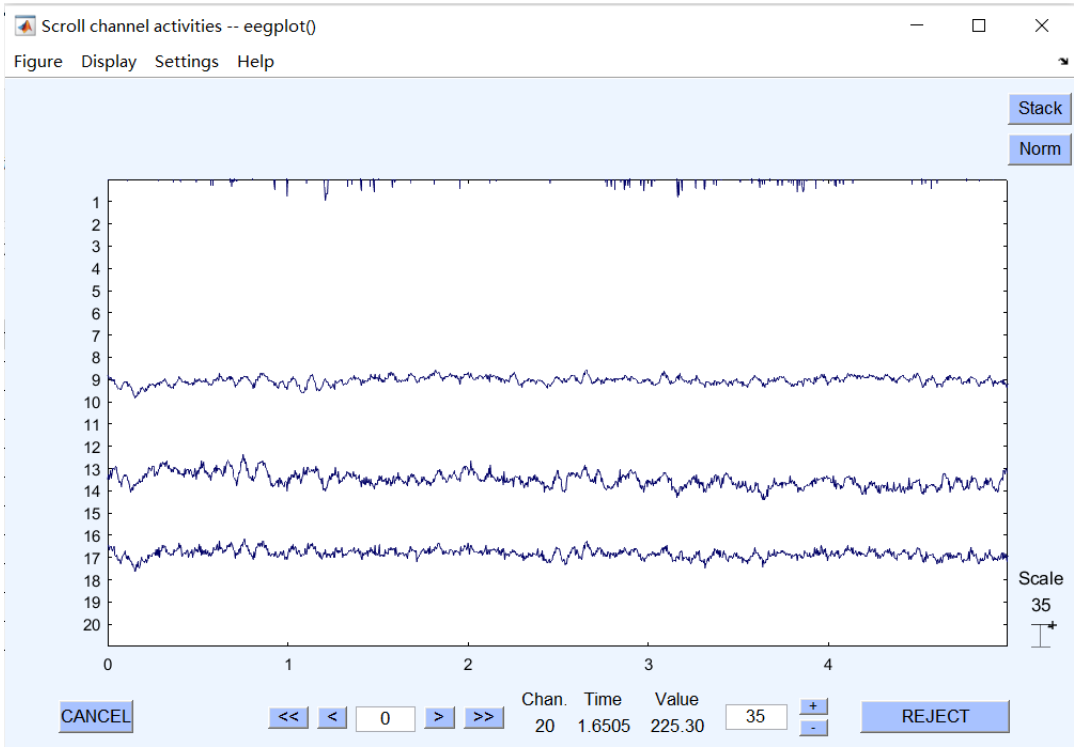


图 3.2.2 滤波前 S1_train_data 中 char06 在 20 个通道上的信号显示

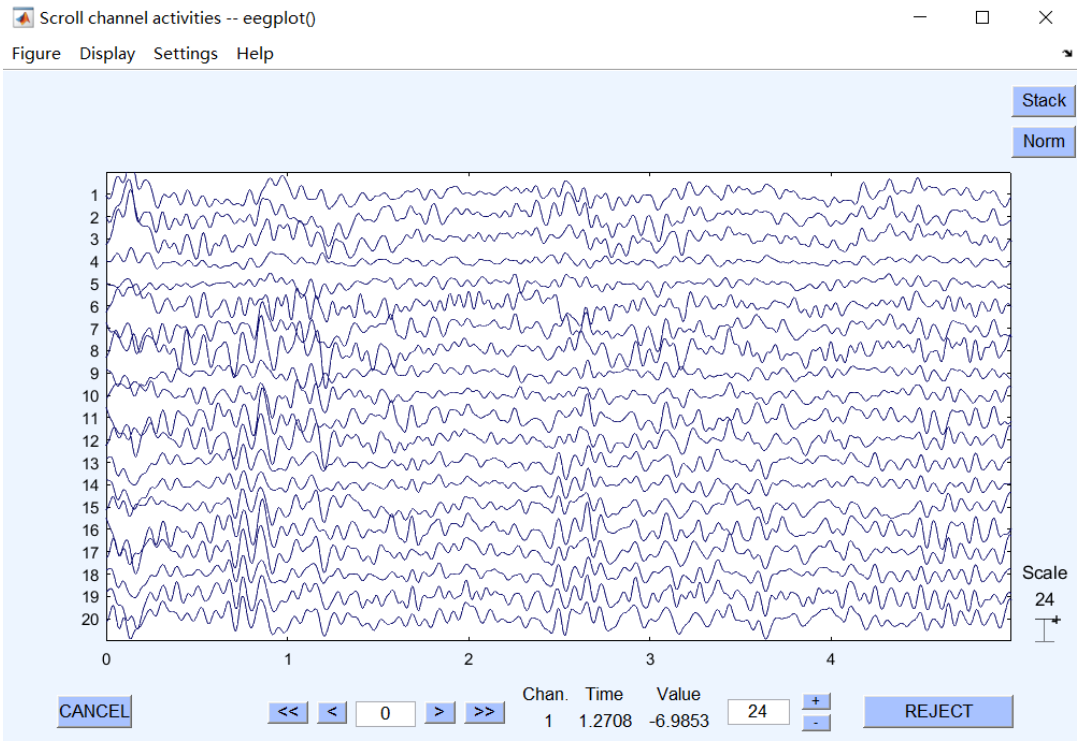


图 3.2.3 滤波后 S1_train_data 中 char06 在 20 个通道上的信号显示

将数据进行滤波处理后再进行目标识别或分类预测的效果明显比初始数据分析效果好很多，因此我们认为眼电、心电等信号干扰会影响脑电信号的分析，并且滤波处理是数据预处理的一项关键步骤。

3.3 不平衡样本的重采样

在训练二分类模型时，正样本：负样本的比例为 1：5，即正负样本数不均衡，如果直接采用不均衡的样本集来进行训练学习，分类器简单地将所有样本都判为负样本就能达到 83.33%的正确率，这样无法实现分类器在正样本和负样本上都有足够的准确率和召回率。针对不平衡数据样本我们主要通过重采样的方法对数据集进行平衡，先随机采样，然后将数据分为训练集和测试集，通过增加正训练样本数和减少负样本数使不平衡样本分布变平衡，从而提高分类器对正样本的识别率。

四、问题一：模型建立与求解

4.1 L2 正则化

在模型训练过程中，由于模型的复杂程度较高，很容易出现过拟合情况（如下图所示[2]），由于模型从训练集的噪音数据中学习了过多的细节，最终导致模型在未知数据上的性能不好，就运算结果来看会出现训练误差很小而测试误差很大的情况。

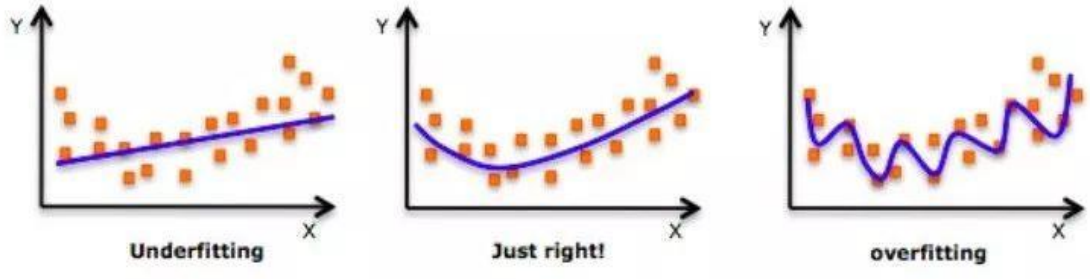


图 4.1.1 三种不同拟合程度下函数分布情况

L2 正则化是指权值向量 w 中各个元素的平方和然后再求平方根，通常表示为 $\|w\|_2$，针对过拟合这一现象，L2 正则化处理可以很好地解决这一问题，主要是通过对最小化经验误差函数添加约束（先验知识），在优化误差函数的时候倾向于选择满足约束的梯度减少的方向，使最终的解倾向于符合先验知识进行调整，以消除奇异性。就 S1 被试者来说，正则化前后随着迭代次数不断增加训练和测试正确率波动情况如图 4.1.2 和图 4.1.3 所示，验证了“过多学习细节会一味追求训练数据的准确率而忽视测试数据准确率”。

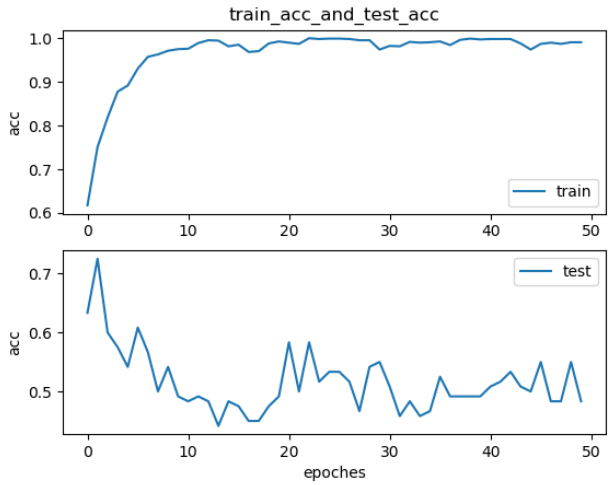


图 4.1.2 正则化前 S1 识别准确率情况

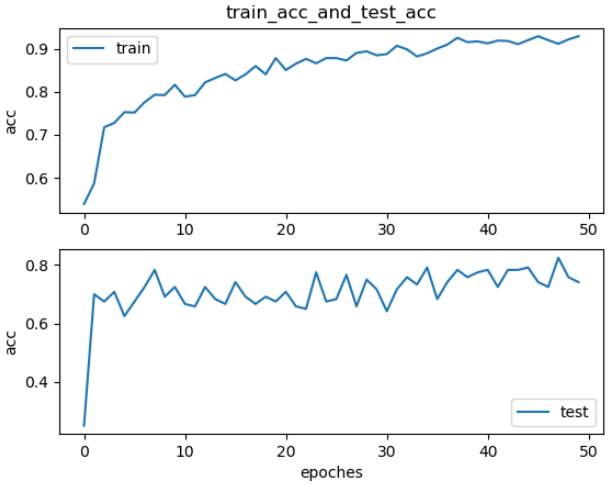


图 4.1.3 正则化后 S1 识别准确率情况

4.2 卷积神经网络方法

关于卷积神经网路（CNN）,它是一种深度学习模型或类似于人工神经网络的多层感知器，常用来分析视觉图像。卷积神经网络是一种带有卷积结构的深度神经网络，卷积结构可以减少深层网络占用的内存量，其三个关键的操作，其一是局部感受野，其二是权值共享，其三是 pooling 层，有效的减少了网络的参数个数，缓解了模型的过拟合问题。

卷积神经网络拓扑结构主要由 5 层组成，每层具体结构如下表 4.2.1 所示，拓扑结构图如图 4.2.2 所示。

表 4.2.1 卷积神经网络各层具体结构

层名称	具体结构
输入层	. 输入初始样本信号（由 20 个通道组成,每通道有 150 个神经元）
空间特征卷积层	. 20 个信号采集通道的一个结合
时间特征卷积层	. 时域下的一个变换及下采样过程，设置为采样间隔时间的一半
全连接层	. 由一个通道组成，其神经元个数为 100，此层为全连接层，将前层个通道提取的特征连接关联起来，综合分析
输出层	. 一个通道，两个神经元，分别用来代表 P300 信号存在与否

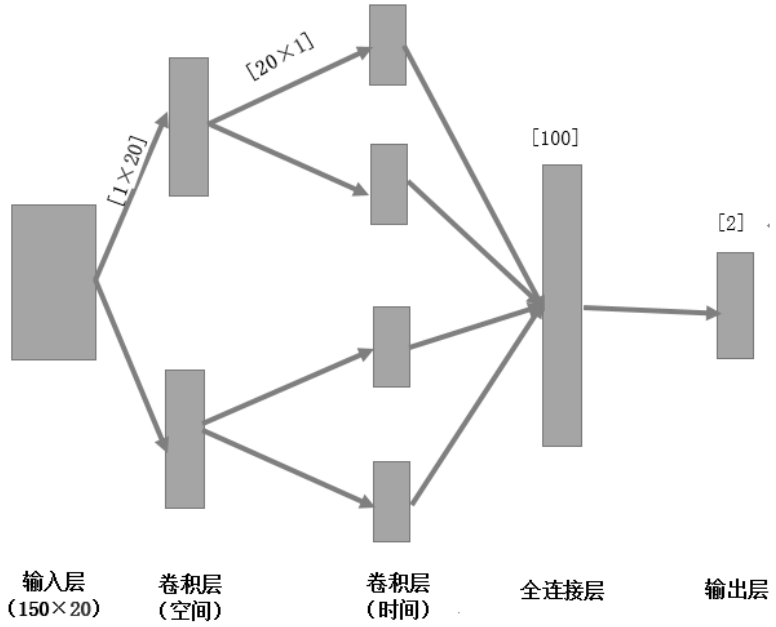


图 4.2.2 问题中卷积神经网络拓扑结构图

整个卷积神经网络在计算过程中可以用矩阵式给出，以单层为例，由于各层的输入输出维度和神经元个数不一样，所以计算的矩阵行列数不相同，但总的计算形式相同，只需要根据不同层的具体情况来对参数进行调整。假设第 i 层有 m 个输入、n 个神经元、n 个输出，则第 i 层的矩阵计算如下所示：

$$O^i=\begin{pmatrix}o_1^i\\ .\\ .\\ .\\ o_n^i\end{pmatrix}=f\left(\begin{pmatrix}w_{11}^i & . & . & . & w_{1m}^i\\ . & . & & & .\\ . & & . & & . .\\ . & & & . & \\ w_{n1}^i & . & . & . & w_{nm}^i\end{pmatrix}\begin{pmatrix}o_1^{i-1}\\ .\\ .\\ .\\ o_m^{i-1}\end{pmatrix}+\begin{pmatrix}b_1^i\\ .\\ .\\ .\\ b_n^i\end{pmatrix}\right) \tag{4.1}$$

O^i 代表第 i 层的输出向量，n 代表第 i 层的神经元数目，第 i-1 层的 m 维输出向量变为第 i 层的输入向量，括号内第一个 $n\times m$ 矩阵代表权重系数矩阵，即 n 个神经元，每个神经元有 m 个权重系数。权重系数矩阵乘以输入向量得到一个 n 维向量，括号里最后一项为 n 维偏置向量，两向量相加后进行激活函数 f 的运算，最终得到第 i 层的输出向量，依次类推，从输入层开始逐层计算，最终得到输出向量。

4.3 随机森林方法

关于随机森林（RF），是用随机的方式建立一个森林，森林里面有很多决策树（决策树多少取决于人为初始设定），不同决策树之间是相互独立的，计算过程中互不干扰，这体现了随机森林算法的集成思想。随机森林的"随机"有两层含义，即样本抽样的随机性和特征抽样的随机性。在得到森林之后，当有一个新的输入样本需要进行分析时，森林中每个决策树分别进行一次运算判断，各自判断输入样本应归为哪一类，最后整合所有决策树的结论，找出被选择较多的类，从而将样本归为那一类，除了用于分类预测外，随机森林还可以用来回归。进行分类预测时算法基本思想可以简化为如下图 4.3.1 所示：

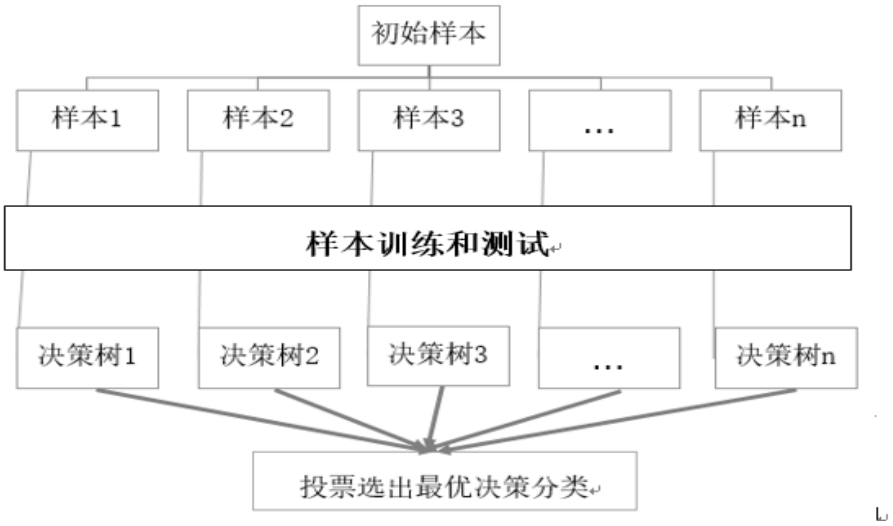


图 4.3.1 随机森林简化分析图

随机森林算法在实现的过程中主要包括三部分：输入、算法、输出，经过一系列转换得到最终的分类或回归结果。具体流程如下表 4.3.2 所示：

表 4.3.2 随机森林实现流程图

随机森林流程	**具体实现**
输入	· 样本集 $D=\{(x_1,y\ ),(x_2,y_2),...(x_m,y_m)\}$ · 弱分类迭代次数 T（决策树数目）
算法	· 对于第 t=1,2，…T 棵树而言： 1. 随机且有放回地从训练集中抽取 m 个训练样本，作为该决策树的训练样本集 D_t 2. 从 D_t 中随机选择一部分特征子集，训练第 t 个决策树模型 3. 每个决策树不剪枝，一直生长到指定的树的深度 · 如果是分类预测，T 个决策树投出最多票数的类别或类别之一为最终类别；如果是回归分析，T 个决策树得到的回归结果进行算术平均得到的值为最终的模型输出
输出	最终的强分类或强回归结果

4.4 目标的分类识别方法设计

待识别字母可以通过查询字母所在 6X6 字母矩阵中行和列的序号获得，因此，识别字母的问题转化为从六行和六列中识别出现 P300 信号的位置的问题。我们在每一轮测试的 12 次闪烁周期中分别判断本周期是否出现 P300 信号，这样，我们就将字母识别问题转化为对信号是否出现的判别问题。

采用经过滤波处理后得到的 1-20Hz 的训练数据 X 和测试数据 C，将数据按照每个字母闪烁周期分段，将训练集中出现 P300 信号的数据段标记为 1，其余数据段标记为 0，将训练数据按照上述预处理过程进行乱序、倍增等处理，将训练数据 X 进一步区分为训练集 A

和测试集 B，隐去测试集的字母标签。

对于训练集 A 的数据，由于从刺激发生到 P300 信号结束一共为 600ms，采样时间间隔为 4ms，故一个待判别信号有 150 个数据，且有 20 个通道。若有该信号中含有 P300 信号，则将其标签设为 1，否则为 0。这样，我们采用[(150X20),(0 或 1)]的形式作为学习的来源。

本题中我们分别采用支持向量机（SVM）、随机森林（RT）、卷积神经网络（CNN）三种方法执行判别功能。由于掌握的数据量较少，常见的训练集和测试集的分配比例是 70%数据设置为训练集，30%数据设置为测试集。但由于掌握的数据量较小，我们采用 80%训练集和 20%测试集的分配比例。

对于支持向量机和随机森林，直接判断信号所在行和列。支持向量机和随机森林的输入形式均为[3000,(0 或 1)]，即将[(150X20),(0 或 1)]的前两维拉直。支持向量机算法的参数设置如下：惩罚系数 C=1，核函数为高斯核函数，核函数参数 gamma=2；随机森林的参数设置如下：树的数量为 400，树的深度为 60，采用交叉熵准则。

对于卷积神经网络，由于采用 TensorFlow 封装的卷积神经网络模块，对[(150X20),(0 或 1)]的输入的（150X20）矩阵的数值单独列为一维，故输入为满足该卷积神经网络模块的四维数据。

4.5 不同方法的合理性分析

我们采用全部 5 轮的数据进行预测。

在测试集 B 上测试各算法的判别效果，纵向对比可以看出卷积神经网络的使用效果要显著强于支持向量机和随机森林，同时横向对比发现，卷积神经网络对 S1 的判别准确性弱于 S2 到 S5，我们猜测这可能和人与人之间的个体差异有关，S1 的脑信号输出一致性较其他人要差一些。因此我们进一步对 S1 的 5 轮信号求平均，获得 1 轮相对稳定的信号，重新采用卷积神经网络进行打分判断，在测试集 B 上的结果显示预测成功率有所提升但依然不高，这可能是由于求平均后只剩下 1 轮数据集，训练集过小。训练 10 次算法并测试求平均后结果如下表 4.5.1 所示：

表 4.5.1 三种不同算法逐轮分析或汇总分析下的识别精准率对比

算法	S1		S2	S3	S4	S5
CNN	S1 求平均前	S1 求平均后	79.0%	79.1%	86.7%	84.9%
	70.2%	80.6%				
SVM	52.7%		54.4%	60.0%	59.8%	62.4%
RT	53.3%		57.6%	63.3%	60.0%	64.7%

三种算法中，卷积神经网络算法的预测成功率最高，因此在接下来对待预测数据集 C 的预测过程采用卷积神经网络进行。

表 4.5.2 卷积神经网络算法对待识别目标精准率的多次结果

轮次	S1 求平均前	S1 求平均后	S2	S3	S4	S5

1	MFG2CNB4A0	ML52CTKXAX	MF52ITKXA	ML52ITKXA	MF52ITKXB0	MF52ITKXA0
2	ML52CBK4AR	ML52CTK4GR	MF52ITK4A	MF52ITKXA	MF52ITKXB0	MF52ITKLA0
3	MF58CNEXA0	ML58CNKXAX	MF52ITK45	MLM2ITKXA	MF52ITKXA0	MF52ITKXA0

由于 S1 的预测结果与其他四组的结果仍然出入较大，且由前述三种算法对 S1 的预测准确度都相对较低，（我们假设该被试者的测试行为有一定的异常），故舍弃 S1 的预测结果。统计 S2 到 S5 的预测结果，每个位置出现频次最高的字母如下表 4.5.3 所示：

表 4.5.3 10 个待识别目标可能值及频数

位置	1	2	3	4	5	6	7	8	9	10
字母	M	F	5	2	I	T	K	X	A	0
频数	12	10	11	12	12	12	12	9	9	6

由于舍弃了 S1 的预测结果，且 S2 和 S3 无第 10 个字母的预测需求，前 9 个字母总频数上限为 12，第 10 个字母的总频数上限为 6。由上表可以看出，预测的一致性非常高。因此，关于 10 个待识别目标我们预测结果为 MF52ITKXA0。

五、问题二：模型建立与求解

首先我们针对单个被试者通道进行了算法设计，有文献显示 P300 脑电信号具有较强的个体差异性，而且不同时间段或刺激发生情况下同一个体的 P300 脑电信号也会有一定的差异[3]，因此找出各个受试者所适合的信号采集通道需要因人而异进行分析，仅仅依靠大脑生理结构或文献研究结论不一定能找出具体受试者最适合的信号通道集合。我们在进行问题分析的过程中，利用卷积神经网络进行分析，优化通道选择算法，找出 S1-S5 被试者各自的最优通道名称组合，然后找出适合所有被试者的一组最优通道组合。

5.1 L1 正则化

L1 正则化是指权值向量 w 中各个元素的绝对值之和，通常表示为 $\|w\|_1$。不同于问题 1 中涉及到的 L2 正则化，L1 正则化可以产生稀疏权值矩阵，即产生一个稀疏模型，可以用于特征选择[2]。

5.2 单个被试者通道选择算法设计

针对本问题样本数据集设计的卷积神经网络如图 4.2.2 所示，第一层卷积层是的卷积核尺寸是 1×20，该层共有 10 个卷积核。滤波之后信号的均值为 0（可以认为这些信号振幅相近相近），将 10 个卷积核在第 i 通道上的权重 weight 相加得到第 i 个通道的总权重。权重越大在一定程度上说明该通道越重要。对该卷积层进行 L1 正则化，在保证训练效果的前提下（在测试集正确率 70%以上），通过调节惩罚因子，使矩阵尽量稀疏。
S1-S5 被试者各自 20 个通道权重值直方图如下图 5.3.1-5.3.5 所示：

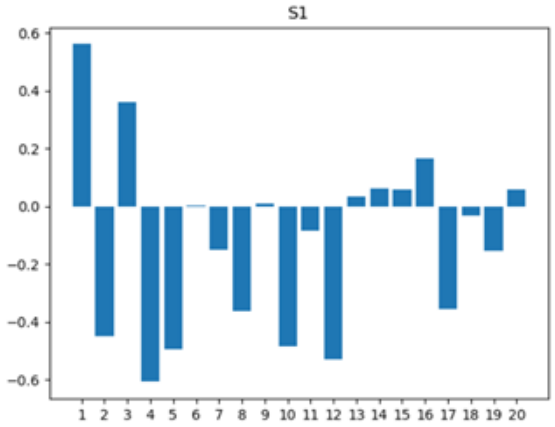


图 5.2.1 S1 试者 20 个通道权重值直方图

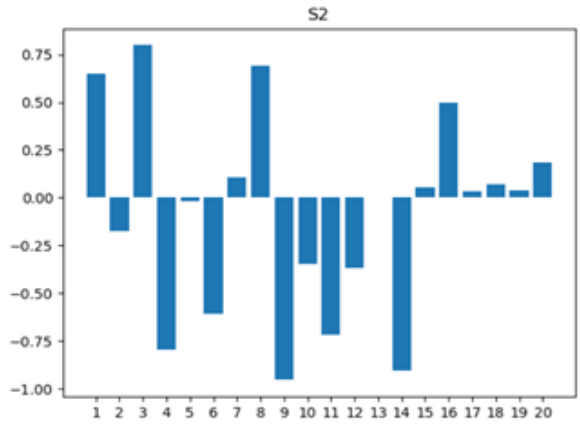


图 5.2.2 S2 试者 20 个通道权重值直方图

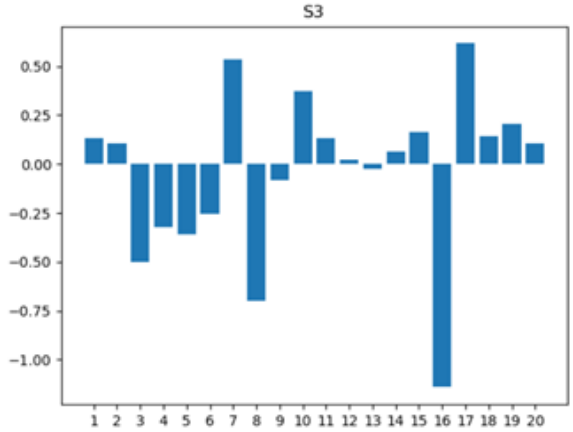


图 5.2.3 S3 试者 20 个通道权重值直方图

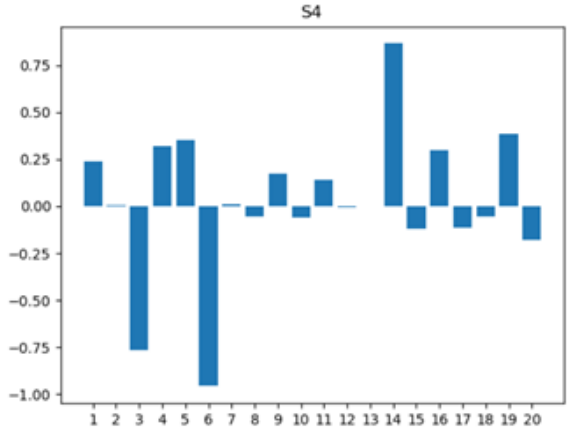


图 5.2.4 S4 试者 20 个通道权重值直方图

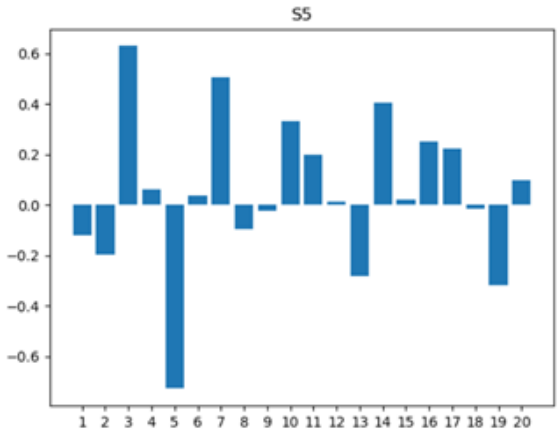


图 5.2.5 S5 试者 20 个通道权重值直方图

依据“权重值越大，通道的重要性越高”这一原则，就 S1-S5 不同被试者各自较优的通道进行筛选，并且保证通道数目在 10-20 之间，筛选结果如下表 5.2.7 所示，各通道选择标识表如下表 5.2.8 所示：

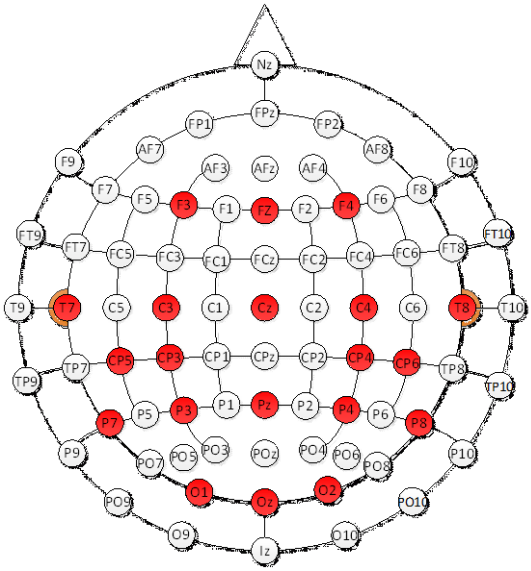


图 5.2.6 脑电信号采集通道图

标识符	通道名称	标识符	通道名称
1	Fz	11	CP5
2	F3	12	CP6
3	F4	13	Pz
4	Cz	14	P3
5	C3	15	P4
6	C4	16	P7
7	T7	17	P8
8	T8	18	Oz
9	CP3	19	O1
10	CP4	20	O2

图 5.2.7 采集通道的标识符

表 5.2.8 S1-S5 被试者适合通道举例

被试者	适合通道（仅用通道标识符表示）
S1	1, 2, 3, 4, 5, 7, 8, 10, 12, 16, 17, 19
S2	1, 3, 4, 6, 8, 9, 10, 11, 12, 14, 16
S3	3, 4, 5, 6, 7, 8, 10, 16, 17, 19
S4	1, 3, 4, 5, 6, 9, 11, 14, 16, 19
S5	1, 2, 3, 5, 7, 10, 11, 13, 14, 19

表 5.2.9 S1-S5 被试者各通道选择标识表

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
S1	■	■	■	■	■		■	■		■		■				■	■		■	
S2	■		■	■		■		■	■	■	■	■		■		■				
S3			■	■	■	■	■	■		■						■	■		■	
S4	■		■	■	■	■			■		■			■		■			■	
S5	■	■	■		■		■			■	■		■	■					■	

在找出不同被试者各自适合的信号通道后，我们又对问题一中待识别目标进行了重新预测，从结果出发检验通道选择的合理性，S1-S5 被试者样本训练和测试的精准率如下图 5.2.10-5.2.14 所示，我们发现训练集的精准率随着学习次数的不断增加，并且精准率逐渐接近于 1，测试集的精准率随着学习次数的不断增加有一定的波动，但基本都接近 80%左右，相较于通道筛选前的预测效果有所增强。

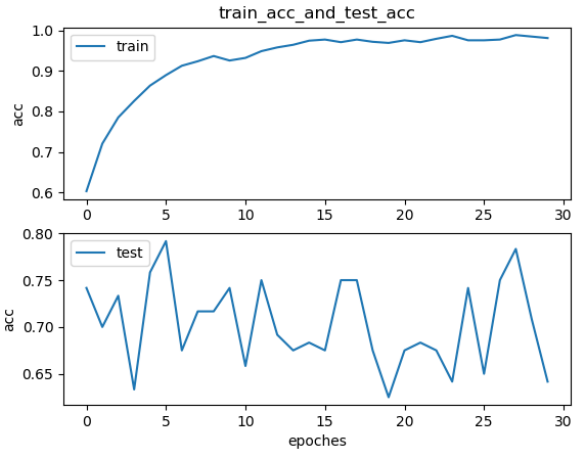


图 5.2.10 S1 样本训练和测试精确率

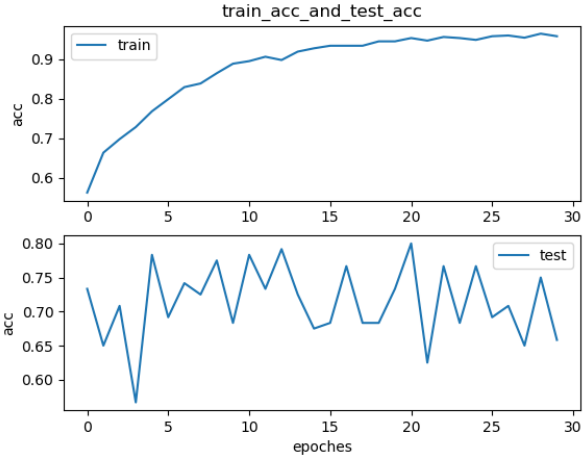


图 5.2.11 S2 样本训练和测试精确率

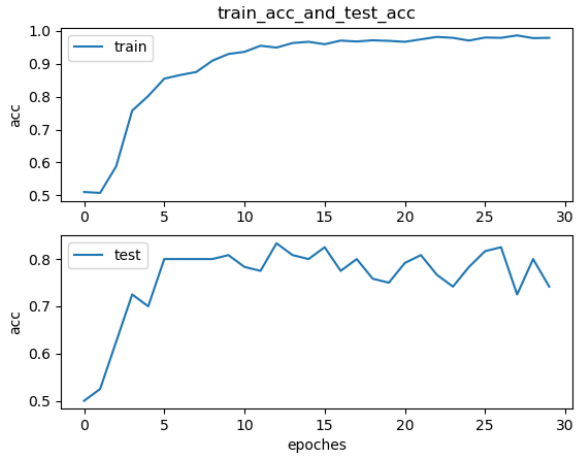


图 5.2.12 S3 样本训练和测试精确率

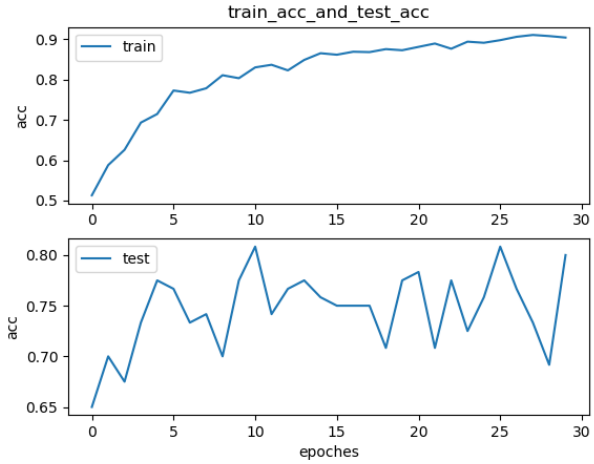


图 5.2.13 S4 样本训练和测试精确率

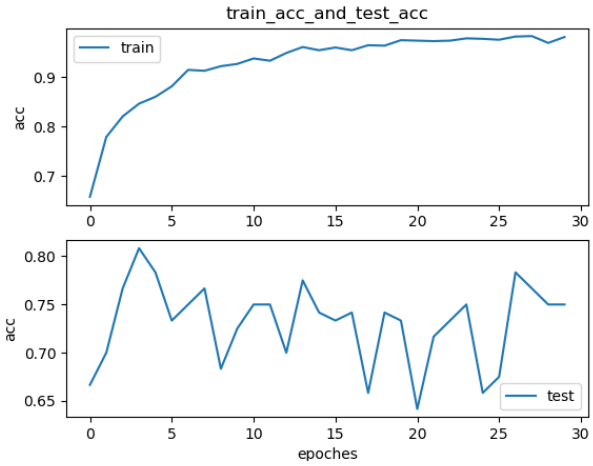


图 5.2.14 S5 样本训练和测试精确率

5.3 所有被试者最优通道组合设计

针对待识别目标，S1-S5 被试者的测试集样本预测得到的其中一次结果如下表所示：

表 5.3.1 S1-S5 被试者的测试集样本预测待识别目标结果

被试者	S1	S2	S4	S5
预测结果	MF52CNA4AX	GF52ITKXM	MF52UTKXAO	MF52ITKXAO

在实验过程中，我们发现 S4 和 S5 样本分类正确的概率更高，可以认为 S4 和 S5 的样本数据集噪音更少。我们假设人类的脑电波特征是近似的，那么干扰少的数据提取出来的信息将更可信，因此在制定所有被试都较适用的一组最优通道名称组合时，我们考虑如下情况：1）将 S4 和 S5 选中的通道全部纳入选择，2）如果一个通道被 3 个以上的被测者选中，那么将该通道纳入选择。基于这种判断机制，最后选出所有被试都较适用的一组最优通道名称组合（一共 15 个通道）：**1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 16, 19**。

进一步以这 15 个通道为输入，重新对所有模型进行训练，S1-S5 样本训练和测试精确率如下图 5.3.2-5.3.6 所示：

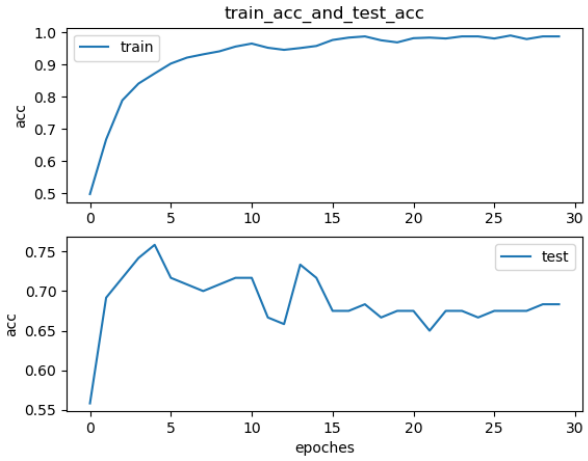


图 5.3.2 基于最优通道 S1 精确率

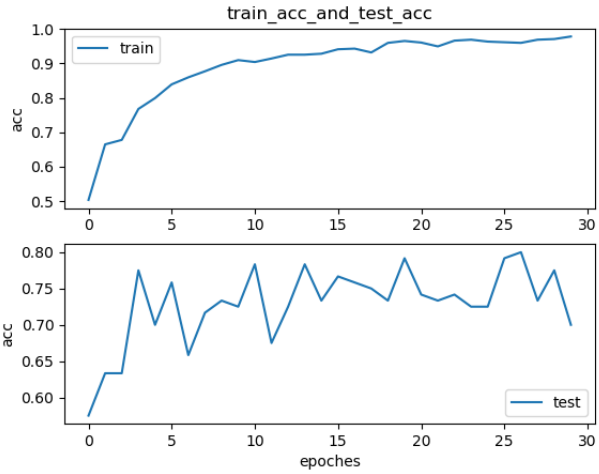


图 5.3.3 基于最优通道 S2 精确率

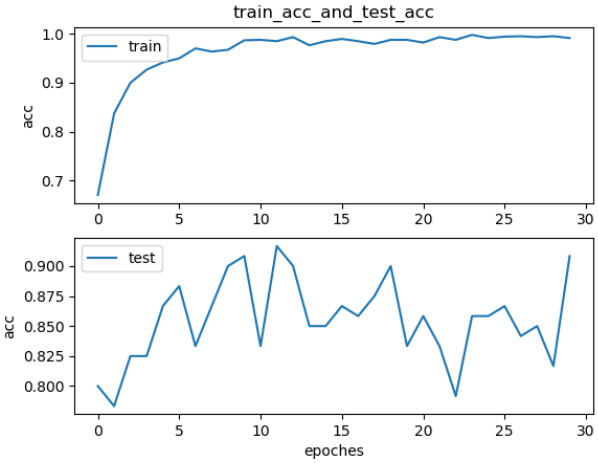


图 5.3.4 基于最优通道 S3 精确率

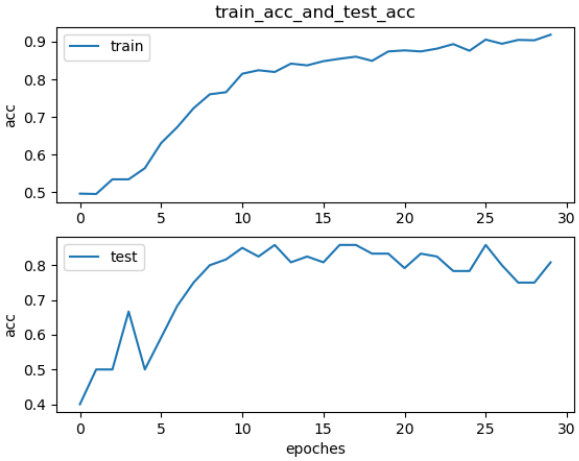


图 5.3.5 基于最优通道 S4 精确率

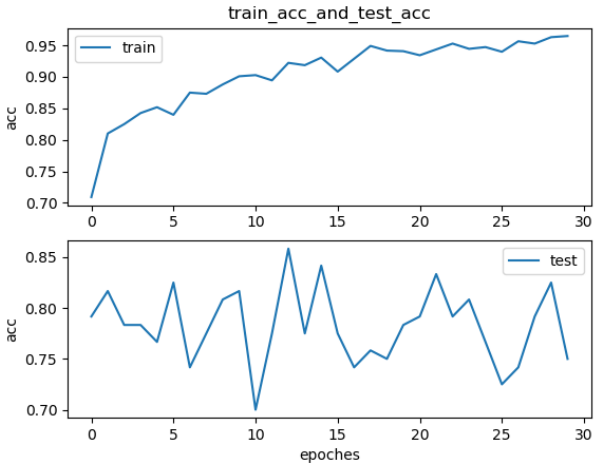


图 5.3.6 基于最优通道 S5 精确率

针对待识别目标，S1-S5 被试者的测试集样本预测得到的结果如下表所示，由于问题二给出前五个待识别目标分别为 M、F、5、2、I，结合下表结果对比发现，受试者 S3、S4、S5 全部识别准确，受试者 S2 识别准确率达 80%，受试者 S1 由于数据存在一定的不稳定，预测准确率不高，总体来说最优通道选择组合比较合理：

表 5.3.7 S1-S5 被试者的测试集样本预测待识别目标结果

被试者	S1	S2	S3	S4	S5
预测结果	MF58CNBLAO	MF52ITKXM	ML52ITKTA	MF52ITKXAO	MF52ITKXAO

六、问题三：模型建立与求解

针对本题同时存在有标签样本和无标签样本的情况，采用半监督学习进行研究探讨。关于半监督学习，结合了有监督学习和无监督学习，其主要思想是基于数据分布的模型假设，这样与有标记样本独立同分布采样的未标记样本，即使未直接包含标记信息，也可利用其数据分布特征与有标记样本相联系。具体来说，利用少量的已标注数据进行指导并预测未标记数据的标记，并合并到标记数据集中去，从而提高学习器的性能[5]。

6.1 标签传播算法

在问题三的求解过程中，采用了半监督学习中的标签传播算法。它的基本思想是给定一个数据集，我们可将其映射为一个图，数据集中的每个样本对应于图中的一个结点，如果两个样本之间的相似度（相关性）很高，则认为对应的结点之间存在一条边，边的强度正比于样本之间的相似度（相关性）[6]。将同一分布范围的已标记结点和未标记结点视为拥有相同的标签，从而将已标记结点的标签信息传播给附近的未标记结点。在每一次迭代过程中，已标记结点的标签保持不变，未标记结点根据相邻结点的标签更新自己的标签。邻近程度由结点的相似程度决定，邻近程度越高，对未标记结点的标签更新的影响权重越大。在不断迭代的过程当中，未标记结点的标签趋向于稳定，从而实现对未标记结点的标签传递。算法的基本描述如下表 6.1.1 所示：

表 6.1.1 标签传播算法步骤及具体实现

算法步骤	**具体实现**
输入	u 个未标记数据 l 个标记的数据及其标签
第一步	初始化，利用权重公式来计算每条边的权重ω_{ij}，得到数据间的相似度
第二步	根据得到的权重ω_{ij}，计算节点 j 到 i 的传播概率T_{ij}
第三步	定义一个(l+u)*C 的矩阵
第四步	每个节点按传播概率把它周围节点传播的标注值按权重相加，并更新到自己的概率分布
第五步	限定已标注的数据，把已标注的数据的概率分布重新赋值为初始值，然后重复步骤四，直至收敛[5]
输出	已标记好的 u 个未标记数据

由于分类器的性能好坏直接取决于对未标记数据标记的准确程度，准确程度低的标记会对分类器的性能产生负面影响。因此，在实际应用标签传播算法的过程中，我们对未标记数据的标签预测结果进行评价，根据评价结果的好坏进行接受或拒绝，从而保证算法效果的正面性。

6.2 自适应半监督学习

在进行标签传播算法实现的过程中会出现不收敛情形，学习效果不是很好，因此我们尝试在卷积神经网络（CNN）的基础上采用自适应半监督学习算法。

6.2.1 算法实现步骤

具体实现步骤如下所示：

第一步：利用带标签样本训练卷积神经网络这一分类器；

第二步：卷积神经网络的 softmax 层输出的是标签是否为 1 或 0 的概率，我们使用训练后的卷积神经网络分类器对无标签样本进行判断，如果该无标签样本判断为 1 的概率大于某一阈值，我们认为这一分类是可信的，将其标签设置为 1，加入有标签的样本集合。如果判断为 0 亦然。同样的如果判断为 1 或 0 的概率均不超过这一阈值，则拒绝该样本，继续置于无标签样本集合；

第三步：如果新增的有标签样本数量为 0，则停止计算，输出有标签样本集。当新增有标签样本数量不为 0 时需要用扩容后的带标签样本集训练卷积神经网络，并返回第二步继续分析。

6.2.2 算法参数设定

首先需将原始数据集划分为有标签数据和无标签数据，如果有标签数据集（训练集）

的规模太小，则无法学到足够的有用特征，分类过程陷入局部最优。经过测试，当有标签数据集的比例低于 0.4 时，分类器的预测准确率很低，而题目要求有标签数据集不宜过大，故在此我们将有标签数据集和无标签数据集的比例设置为 0.4 和 0.6。

关于接受阈值的设置，如果接受标签为 1 或 0 的阈值过低，那么就会接受大量对后续判断无益甚至有害的样本，如果如果接受标签为 1 或 0 的阈值过高，那么就会只接受距离有标签样本非常近的样本，而拒绝一些原本可以对后续判断有益的样本。因此，该阈值的设置同样比较重要，经过测试，决定采用 0.8 作为接受阈值。

6.2.3 算法识别结果分析

我们就多轮测试中的一次结果进行展示，针对不同受试者就采用自适应半监督学习算法前后进行对比，对比情况如表 6.2.2 所示：

表 6.2.2 采用自适应半监督学习算法前后预测结果对比

受试者	采用自适应半监督学习算法前后	待识别目标预测字母
S1	前	MF5OUNKOGO
	后	AF5YCBK4GO
S2	前	GF52JTKV5
	后	GF52ITKXA
S3	前	MFM2ITKXA
	后	ML52ITKXA
S4	前	MF52ITKXBO
	后	MF52ITKJAO
S5	前	MF52ITKOAO
	后	MF52ITKXAO

实验结果显示，采用自适应半监督学习算法后预测准确率有一定提升。如对于 S3，只采用比例为 0.4 的有标签样本训练神经网络分类器的预测准确率为 80.9%；采用上述自适应半监督学习算法，同时利用有标签样本和可信的无标签样本训练神经网络分类器的预测准确率为 83.4%，相比于前者的预测准确率有一定提升。对于前文中预测准确率不高的 S1，其预测准确率也有所提高。

另一方面，采用监督和半监督算法来预测不同受试者识别效果，不同受试者效果对比图如下所示，我们发现后者的预测稳定性有大幅提升，只采用比例为 0.4 的有标签样本训练神经网络分类器的预测准确率变化幅度很大，这可能和训练集仍然较小有关。而时利用有标签样本和可信的无标签样本训练神经网络分类器的预测准确率则相对更加稳定，这对于实际应用有较大益处，体现了本方法的效果。

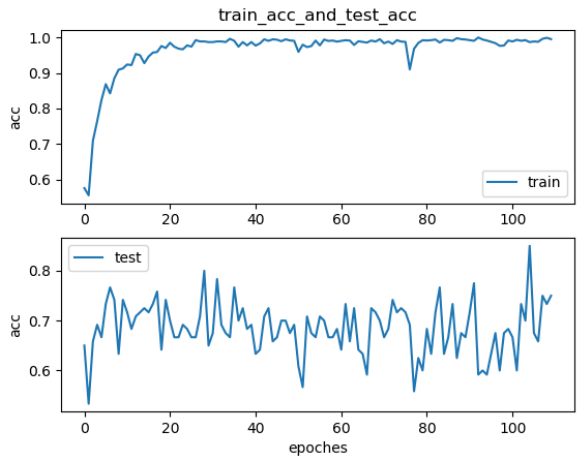


图 6.2.3 采用半监督学习算法 S1 准确率

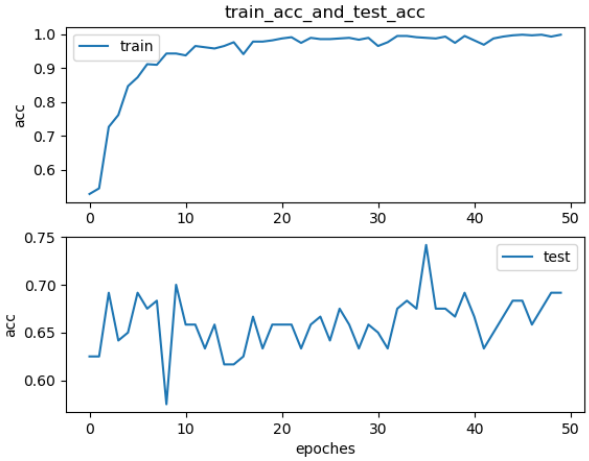


图 6.2.4 采用监督学习算法 S1 准确率

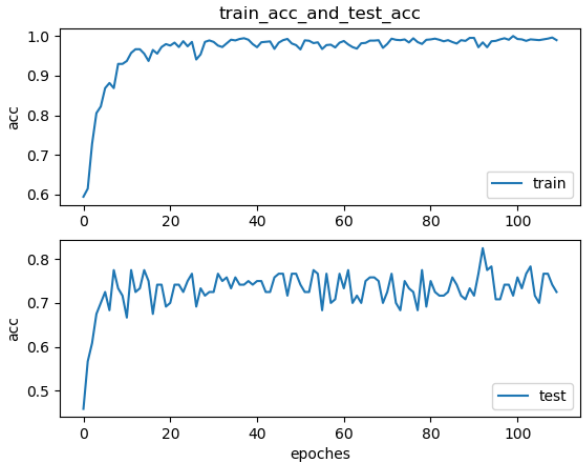


图 6.2.5 采用半监督学习算法 S2 准确率

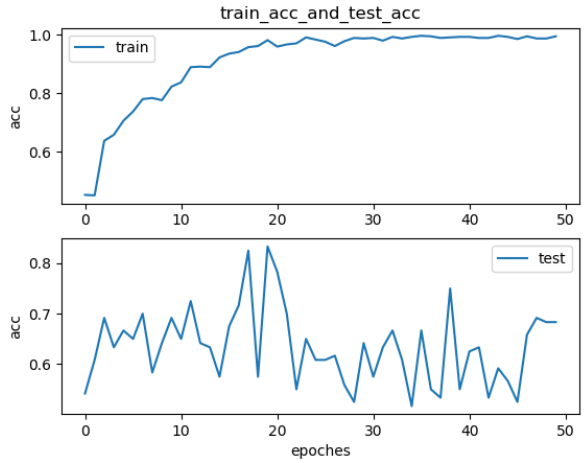


图 6.2.6 采用监督学习算法 S2 准确率

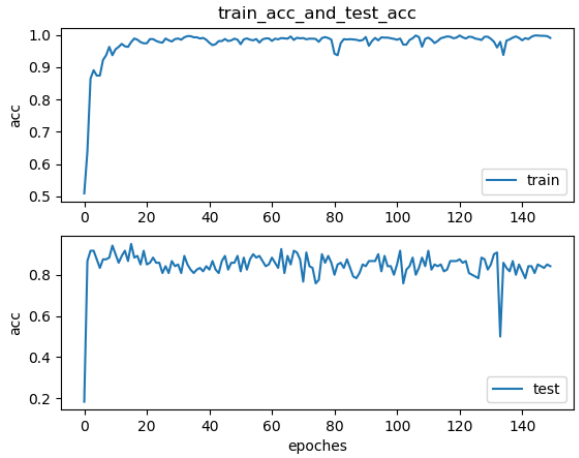


图 6.2.7 采用半监督学习算法 S3 准确率

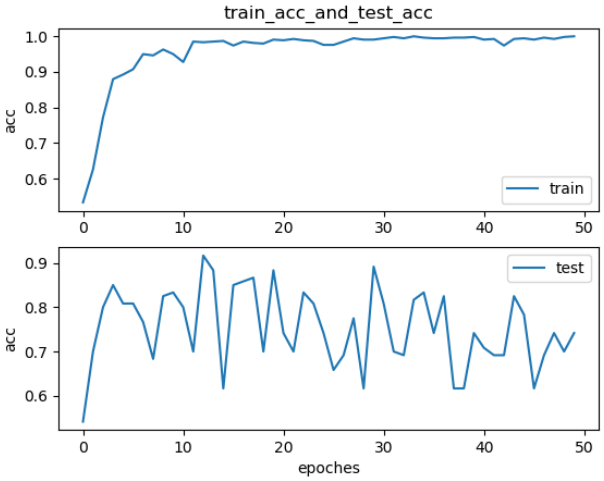


图 6.2.8 采用监督学习算法 S3 准确率

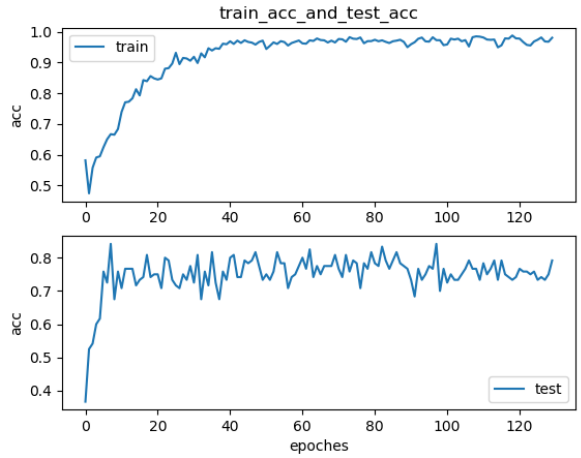


图 6.2.9 采用半监督学习算法 S4 准确率

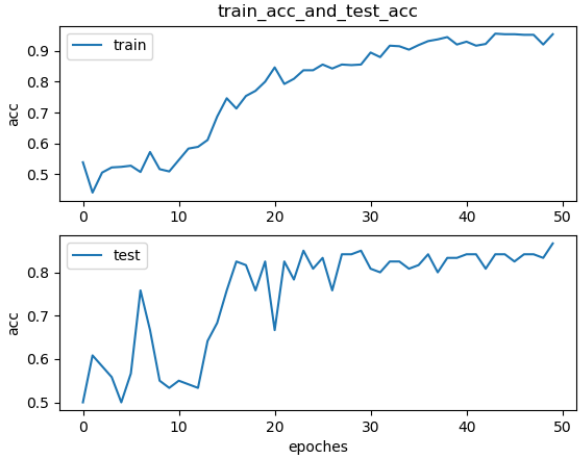


图 6.2.10 采用监督学习算法 S4 准确率

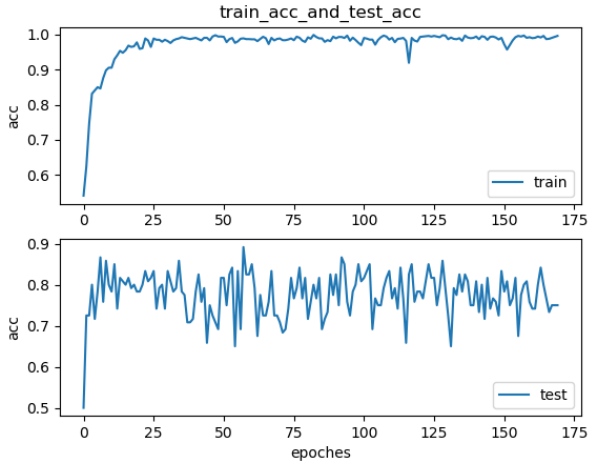


图 6.2.11 采用半监督学习算法 S5 准确率

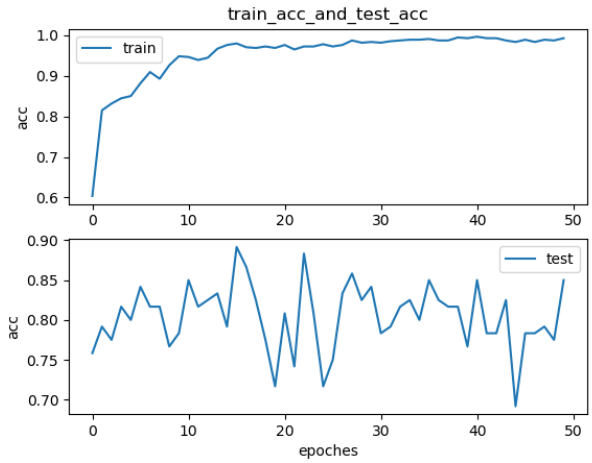


图 6.2.12 采用监督学习算法 S5 准确率

接下来进入预测阶段，上表对前五个字母 MF52I 的预测准确程度如下表：

表 6.2.13 采用自适应半监督学习算法前后前五个字母预测正确的数量

受试者	采用自适应半监督学习算法前后	前五个字母预测正确的数量
S1	前	3
	后	2
S2	前	3
	后	4
S3	前	4
	后	4
S4	前	5
	后	5
S5	前	5

	后	5

我们对前五个字母预测正确的数量大于等于 4 的预测字母组合，即 S2 后，S3 前后，S4 前后和 S5 前后进行统计，统计后五个字母的每个位置出现频次最高的字母如下表：

表 6.2.14 后 5 个待识别字母的预测及其频数

位置	6	7	8	9	10
字母	T	K	X	A	0
频数	7	7	5	6	4

由于舍弃了 S1 的全部预测结果和 S2 的监督学习预测结果，且 S2 和 S3 无第 10 个字母的预测需求，第 6 到 9 位字母频数上限各为 7，第 10 个字母的频数上限为 4。由上表可以看出，预测的一致性比较高。因此，预测后 5 个字母结果为 TKXA0。

七、问题四：模型建立与求解

为更好的设计出睡眠分期预测模型，在求解的过程中主要用到了随机森林和支持向量机两种方法进行预测，同时每一种方法就训练样本占总样本比例进行不断的调整，以求找出最合适精准率最高的分配比例。

7.1 支持向量机方法

关于支持向量机（SVM），它是建立在统计学习理论基础上的一种数据挖掘方法，用来处理回归问题(时间序列分析)和模式识别(分类问题、判别分析)等诸多问题，在预测和综合评价等领域和学科有着十分广泛的应用。核心思想是寻找一个满足分类要求的最优分类超平面，使得该超平面在保证分类精度的同时，使超平面两侧的空白区域最大化。样本中距离超平面最近的一些点组成的向量叫做支持向量，判断分类效果好坏的依据是不同样本都能归属到最适合的类别中，同时各类样本点到超平面的距离最远，即不同类的样本点之间距离最大。

间隔超平面所在位置点可以用下面方程来描述：

$$w^T x + b = 0 \tag{7.1}$$

在最常见的二维空间内点 (x, y) 到直线 $Ax + By + C = 0$ 的距离 d_1 是：

$$d_1 = \frac{|Ax + By + C|}{\sqrt{A^2 + B^2}} \tag{7.2}$$

扩展到 n 维空间后，点 $x = (x_1, x_2 ... x_n)$ 到直线 $w^T x + b = 0$ 的距离 d_2 为：

$$d_2 = \frac{|w^T x + b|}{\|w\|} \tag{7.3}$$

其中 $\|w\| = \sqrt{w_1^2 + w_2^2 + ... w_n^2}$

如下图所示，w和b 是人为设计的训练参数，形成了两条间隔线 $w^T x + b = 1$和$w^T x + b = -1$，正确的分类情况应该是所有样本点正确分布在处于自己的那一侧，并且不在两条间隔线之间出现，对样本中任意一点 x_i 应有以下约束：

$$\begin{cases} w^T x_i + b \geq 1 & , y_i = 1 \\ w^T x_i + b \leq -1 & , y_i = -1 \end{cases} \tag{7.4}$$

已知点 x_i 到超平面 (w,b) 的距离为 d，如下图所示，只有处于间隔线上的点对间隔线有影响，比如下图中的一个正点和两个负点，对于间隔线上的点来说 $|w^T x + b| = 1$，此时距离可简化成 $d_3 = \frac{1}{\|w\|}$，支持向量机方法的优化目标为在约束条件下求得 d_3 最大，经过一系列线性和对偶转换，最终转换为求以下规划问题：

$$\begin{aligned} & Min_{w,b} && \frac{1}{2}\|w\|^2 \\ & s.t. && y_i(w^T x_i + b) \geq 1 \ , \ i = 1,2,...,m \end{aligned} \tag{7.5}$$

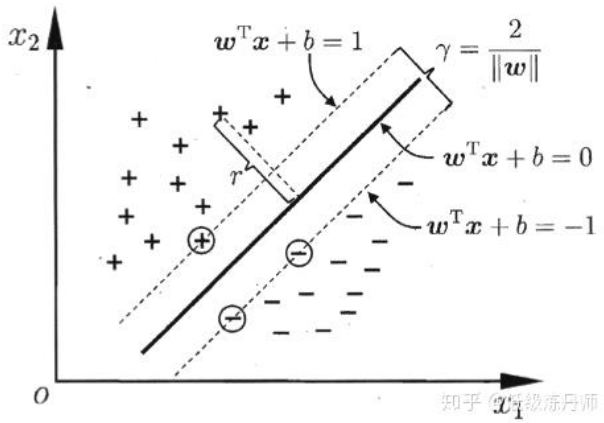


图 7.2.1 线性可分二维平面情况

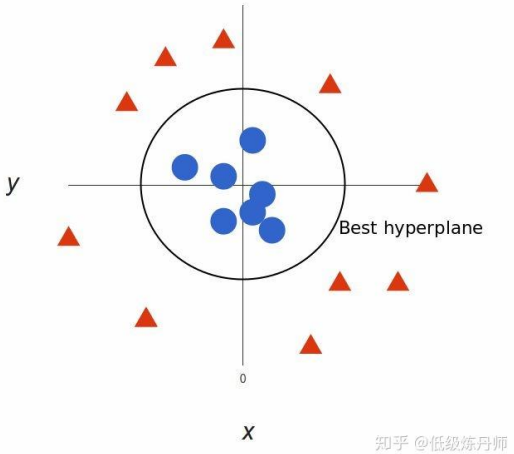


图 7.2.2 完全线性不可分平面情况

如图所示，当数据完全线性不可分时，即无法找到一条间隔直线或间隔超平面，这时需要将数据投影到高维线性可分空间，在新映射空间内寻找超平面，通过间隔最大化的方式，学习得到支持向量机，即非线性 SVM。假设点 x 为初始样本点，$\phi(x)$ 为 x 映射到高维空间后得到的新向量，此时的分割超平面表示为 $f(x) = w\phi(x) + b$，优化问题经一系列线性和对偶转换，最终转换为求以下规划问题：

$$\begin{aligned} & \underset{\lambda}{Min} && \left[\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (\phi(x_i) \bullet \phi(x_j)) - \sum_{j=1}^{n} \lambda_i \right] \\ & s.t. && \sum_{i=1}^{n} \lambda_i y_i = 0, \quad \lambda_i \geq 0, \quad C - \lambda_i - \mu_i = 0 \end{aligned} \tag{7.6}$$

线性 SVM 和非线性 SVM 唯一的不同在于初始样本点 (x_i, y_i) 和映射点 $(\phi(x_i, y_i)$ 的差异，但求解过程中非线性 SVM 更为复杂，往往通过引入核函数降低计算工作量。

7.2 睡眠分期预测模型选择

在进行预测的过程中我们选择了支持向量机和神经网络两种方法进行尝试，由于神经网络在训练的过程中需要的样本数量较大，针对少样本训练集得到的预测能力较差，如下图 7.1.1 所示，测试集的预测准确率随着迭代次数的增加不断上下波动，且维持在 [0.5,0.675] 的范围内变化，准确率较低，不太适合进行少样本预测。而支持向量机方法在少样本训练测试中的效果较为明显，因此我们选择使用支持向量机方法来设计睡眠分期

预测模型。

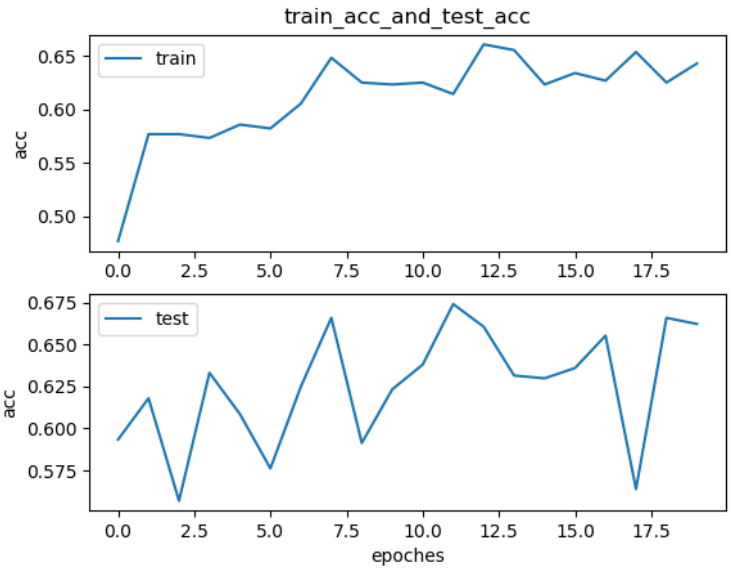


图 7.1.1 神经网络方法预测睡眠分期模型准确率情况

7.3 训练和测试数据的选择

较为理想的训练和测试数据比例应该是以尽可能少的样本对睡眠状态分期预测的较为精准，但在实际操作过程中我们发现很难实现这一平衡，只能保证在较高的预测准确率条件下选择尽可能少的样本。

7.3.1 选取方式

在选取样本前，我们对不同时期的样本数进行了统计，结果如下表所示：

表 7.3.1 不同睡眠状态期样本量情况

睡眠状态	清醒期	睡眠 I 期	睡眠 II 期	深睡眠期	快速眼动期
样本量	634	563	605	603	600

为保证样本数的一致性，每个睡眠状态期选取 112（560*0.2）个样本进行分析。另外样本在抽取之前顺序打乱弄混以实现随机抽样，然后在不同状态时期随机选取 112 个样本进行训练预测。

7.3.2 分配比例

基于支持向量机（SVM）方法的训练和学习，为了能够较为直观准确的找出训练集比例小但测试准确率高这一均衡点，将测试集准确率随训练集比例变化情况用折线图表示出来，如下图 7.3.2 所示，随着训练集的使用比例增加，在测试集上的分类准确率整体呈上升趋势，具体来讲，训练集使用比例在 0.1 到 0.2 和 0.8 到 0.9 两区间内上升趋势明显，而比例在 0.2-0.8 区间范围内趋势平缓（测试集正确率随训练样本增加不明显），由此我们认为使用训练集比例为 0.2 时，可保证在使用较少训练集数据的前提下获得良好的分类效果。另外我们就算法参数进行了不断的调试，最终确定了惩罚系数 C=1，核函数为高斯核函数，核函数参数 gamma=2，然后选取 0.2 的训练集比例能够获得较高的预测准确率。

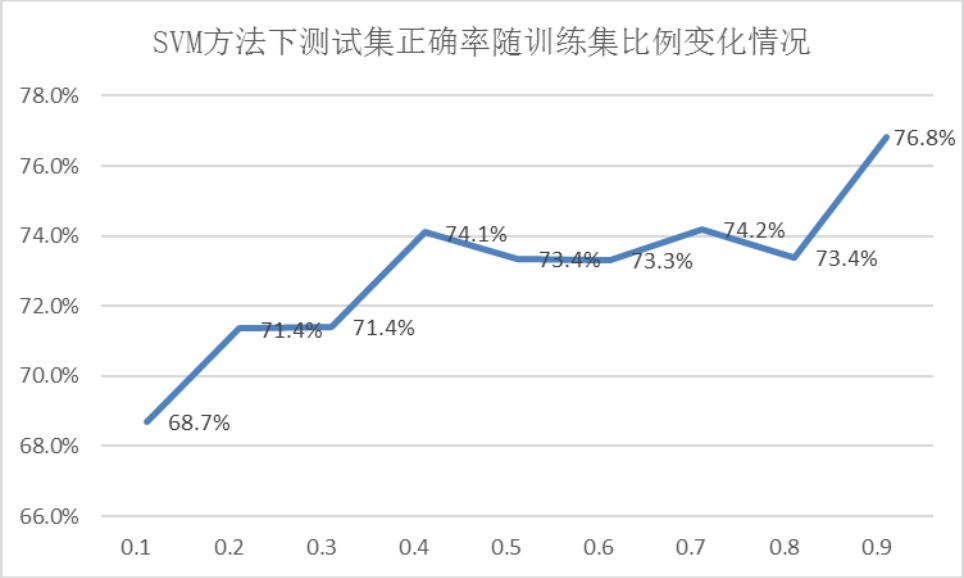


图 7.3.2 SVM 方法下测试集正确率随训练集比例变化情况

7.4 结合分类性能分析模型预测效果

在判断分类性能好坏时我们一般关心样本归为正确分类的概率、样本归为错误分类的概率是多大，需要参考一系列的分类性能指标，比如召回率、查准率、混淆矩阵等，在多分类问题中比较常用的是混淆矩阵，具体包含内容如下表 7.4.1 所示[7]。通过判断四类情况发生的概率值大小来更好的评价模型分类性能好坏。

表 7.4.1 混淆矩阵内容

		预测情况		合计
		正	负	
实际情况	正	实际为正预测为正概率	实际为负预测为正概率	实际为正概率
	负	实际为负预测为正概率	实际为负预测为负概率	实际为负概率
合计		预测为正概率	预测为负概率	1

八、参考文献

[1]计瑜. 基于独立分量分析的 P300 脑电信号处理算法研究[D]. 浙江大学.2013 年.
[2]搜狐网，深度学习中的正则化技术(附 Python 代码)：
https://www.sohu.com/a/ 233135994 197042，2020-09-20.
[3] Rakotomamonjy A,Guigue V.BCI competition Ⅲ:dataset Ⅱ- ensemble of SVMs for BCI P300 speller[J].IEEE Transactions on Biomedical Engineering,2008,55(3): 1147-1154.
[4]Microstrong，主成分分析原理详解：
https://zhuanlan.zhihu.com/p/37777074，2020-09-20.
[5]周志华，机器学习，北京：清华大学出版社，300-301，2016.

[6]诗蕊，标签传播算法(Label Propagation Algorithm)：
https://blog.csdn.net/Katherine_hsr/article/details/82343647?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase%EF%BC%8C，2020-09-20.
[7]Dana-Song,分类模型的评价指标--混淆矩阵:
https://blog.csdn.net/shy19890510/article/details/79501582 ，2020-09-20.

附录：Python 代码

代码附录：

```
# ---------------------
# 问题一：svm 和 RF
# ---------------------
import pandas as pd
import numpy as np
import os
import pickle
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.externals import joblib

# 为了减少反复 io，将数据序列号保存至 sri.txt
seril_path = 'sri.txt'
row_col = {'A':(1,7),'B':(1,8),'C':(1,9),'D':(1,10),'E':(1,11),'F':(1,12),
       'G':(2,7),'H':(2,8),'I':(2,9),'J':(2,10),'K':(2,11),'L':(2,12),
       'M':(3,7),'N':(3,8),'O':(3,9),'P':(3,10),'Q':(3,11),'R':(3,12),
       'S':(4,7),'T':(4,8),'U':(4,9),'V':(4,10),'W':(4,11),'X':(4,12),
       'Y':(5,7),'Z':(5,8),'1':(5,9),'2':(5,10),'3':(5,11),'4':(5,12),
       '5':(6,7),'6':(6,8),'7':(6,9),'8':(6,10),'9':(6,11),'0':(6,12)}
rc_dic = [['A','B','C','D','E','F'],
       ['G','H','I','J','K','L'],
       ['M','N','O','P','Q','R'],
       ['S','T','U','V','W','X'],
       ['Y','Z','1','2','3','4'],
       ['5','6','7','8','9','0']]

# 读取序列号之后的数据，x_p 是正例集合（检测到 P300），x_n 是反例集合
with open(seril_path,'rb') as f:
```

```
    x_p,y_p,x_n,y_n = pickle.load(f)

# 将数据乱序
np.random.shuffle(x_p)
np.random.shuffle(x_n)
# 将数据拉直
x_p = [np.array(x).reshape(-1) for x in x_p]
x_n = [np.array(x).reshape(-1) for x in x_n]

# 按 8：2 将训练集拆分，8 用来训练，2 用来判断准确率
a = 96
b = 480

# 由于样本不平衡（正例：反例=1：5），所以将训练集拆分后，再分别把两部分的 x_p 倍增
x_p_train = np.repeat(x_p[0:a],5,axis=0)
y_p_train = np.repeat(y_p[0:a],5,axis=0)
x_p_test = np.repeat(x_p[a:],5,axis=0)
y_p_test = np.repeat(y_p[a:],5,axis=0)
x_train = np.vstack([x_p_train,x_n[0:b]])
y_train = np.hstack([y_p_train,y_n[0:b]])
x_test = np.vstack([x_p_test,x_n[b:]])
y_test = np.hstack([y_p_test,y_n[b:]])


# 训练集乱序
np.random.seed(116)
np.random.shuffle(x_train)
np.random.seed(116)
np.random.shuffle(y_train)

# 以下为随机森林和 svm 的模型，这里将两个模型写在了一起，实际只运行其中一个
# 定义随机森林，树的数量为 400，深度为 60
model                                                                    =
RandomForestClassifier(n_estimators=400,criterion='entropy',max_depth=60)
# 定义支持向量机，惩罚因子 c=1，核函数为高斯核，参数为 2
model = SVC(C = 1, kernel='rbf', gamma=2)

# 模型训练及准确度计算
model = model.fit(x_train,y_train)
y_test_hat = model.predict(x_test)
y_test = y_test.reshape(-1)
print(y_test_hat)
print(y_test)
```

```
result = (y_test == y_test_hat)
acc = np.mean(result)
print("精准度：%.2f%%" % (100 * acc))



# ---------------------
# 问题一：cnn
# ---------------------
#
import pandas as pd
import numpy as np
import os
import pickle
import sklearn
import tensorflow as tf
from                    tensorflow.keras.layers                    import
Dropout,Dense,Conv2D,Flatten,BatchNormalization,Activation
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

seril_path = 'sri.txt'
row_col = {'A':(1,7),'B':(1,8),'C':(1,9),'D':(1,10),'E':(1,11),'F':(1,12),
       'G':(2,7),'H':(2,8),'I':(2,9),'J':(2,10),'K':(2,11),'L':(2,12),
       'M':(3,7),'N':(3,8),'O':(3,9),'P':(3,10),'Q':(3,11),'R':(3,12),
       'S':(4,7),'T':(4,8),'U':(4,9),'V':(4,10),'W':(4,11),'X':(4,12),
       'Y':(5,7),'Z':(5,8),'1':(5,9),'2':(5,10),'3':(5,11),'4':(5,12),
       '5':(6,7),'6':(6,8),'7':(6,9),'8':(6,10),'9':(6,11),'0':(6,12)}
rc_dic = [['A','B','C','D','E','F'],
       ['G','H','I','J','K','L'],
       ['M','N','O','P','Q','R'],
       ['S','T','U','V','W','X'],
       ['Y','Z','1','2','3','4'],
       ['5','6','7','8','9','0']]

# 数据的读入与切分及乱序，与上文相同
with open(seril_path,'rb') as f:
    x_p,y_p,x_n,y_n = pickle.load(f)
x_p = np.array([np.array(x) for x in x_p])
x_n = np.array([np.array(x) for x in x_n])
x_p = x_p.reshape(x_p.shape[0],x_p.shape[1],x_p.shape[2],1)
x_n = x_n.reshape(x_n.shape[0],x_n.shape[1],x_n.shape[2],1)
np.random.shuffle(x_p)
np.random.shuffle(x_n)
```

```
a = 96
b = 480
x_p_train = np.repeat(x_p[0:a],5,axis=0)
y_p_train = np.repeat(y_p[0:a],5,axis=0)
x_p_test = np.repeat(x_p[a:],5,axis=0)
y_p_test = np.repeat(y_p[a:],5,axis=0)
x_train = np.vstack([x_p_train,x_n[0:b]])
y_train = np.hstack([y_p_train,y_n[0:b]])
x_test = np.vstack([x_p_test,x_n[b:]])
y_test = np.hstack([y_p_test,y_n[b:]])
np.random.seed(114)
np.random.shuffle(x_train)
np.random.seed(114)
np.random.shuffle(y_train)


# 定义卷积神经网络的结构，
# 第一层是 1X20 的卷积核，起到了空间特征提取的作用（整合 20 个通道）
# 第二层是 20X1 的卷积核，起到了时间特征提取的作用（在长度为 150 的信号序列上滑动）
# 之后是全连接层和输出层
# 对卷积层进行 0.2 比例的参数舍弃，每次训练只训练 80%参数，同时在全连接层进行 12 正则，防止过拟合
# 对第一个卷积核进行 11 正则，获得稀疏矩阵，方便通道选择
model = tf.keras.Sequential([
    Conv2D(filters=10, kernel_size=(1,20),
        # kernel_regularizer=tf.keras.regularizers.l1(0.004)
        ),
    Dropout(0.2),
    Conv2D(filters=50, kernel_size=(20,1),strides=10),
    BatchNormalization(),
    Activation('relu'),
    Dropout(0.2),
    Flatten(),
    Dense(100,activation = 'relu',
        # kernel_regularizer=tf.keras.regularizers.l2(0.003)
        ),
    Dense(2,activation='softmax',
        kernel_regularizer=tf.keras.regularizers.l2(13)
        )])

# 模型的求解器选择
model.compile(optimizer = tf.keras.optimizers.Adam(0.01),
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
```

```
    metrics='sparse_categorical_accuracy')
# 设置模型保存路径，以供预测
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath='cnn.ckpt',
    save_weights_only=True,
    save_best_only=True)
history = model.fit(x_train,y_train,batch_size=20,epochs = 30,
    validation_data=(x_test,y_test),validation_freq=1,callbacks=[cp_callback])

acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']

# 画出模型的准确率随迭代次数的变化图
# acc 为训练集上的准确率，val_acc 为测试集上的准确率
plt.subplot(2,1,1)
plt.title('train_acc_and_test_acc')
plt.plot(acc,label='train')
# plt.xlabel('epoches')
plt.ylabel('acc')
plt.legend()
plt.subplot(2,1,2)
plt.plot(val_acc,label='test')
plt.xlabel('epoches')
plt.ylabel('acc')
plt.legend()
plt.show()

# ----------------------
# 问题一：预测
# ----------------------
#
import pandas as pd
import numpy as np
import os
import pickle
import sklearn
import tensorflow as tf
from                    tensorflow.keras.layers                    import
Dropout,Dense,Conv2D,Flatten,BatchNormalization,Activation
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

row_col = {'A':(1,7),'B':(1,8),'C':(1,9),'D':(1,10),'E':(1,11),'F':(1,12),
       'G':(2,7),'H':(2,8),'I':(2,9),'J':(2,10),'K':(2,11),'L':(2,12),
```

```
        'M':(3,7),'N':(3,8),'O':(3,9),'P':(3,10),'Q':(3,11),'R':(3,12),
        'S':(4,7),'T':(4,8),'U':(4,9),'V':(4,10),'W':(4,11),'X':(4,12),
        'Y':(5,7),'Z':(5,8),'1':(5,9),'2':(5,10),'3':(5,11),'4':(5,12),
        '5':(6,7),'6':(6,8),'7':(6,9),'8':(6,10),'9':(6,11),'0':(6,12)}
rc_dic = [['A','B','C','D','E','F'],
        ['G','H','I','J','K','L'],
        ['M','N','O','P','Q','R'],
        ['S','T','U','V','W','X'],
        ['Y','Z','1','2','3','4'],
        ['5','6','7','8','9','0']]


# 读取已经序列化的待预测数据集（test）中的数据
with open('predict.txt','rb') as f:
    data_set_total = pickle.load(f)


# 声明一个与已训练模型结构相同的卷积神经网络
model = tf.keras.Sequential([
    Conv2D(filters=10, kernel_size=(1,20),
        # kernel_regularizer=tf.keras.regularizers.l1(0.01)
        ),
    Dropout(0.3),
    Conv2D(filters=50, kernel_size=(20,1),strides=10),
    BatchNormalization(),
    Activation('relu'),
    Dropout(0.3),
    Flatten(),
    Dense(100,activation = 'relu',
        kernel_regularizer=tf.keras.regularizers.l2(0.0001)
        ),
    Dense(2,activation='softmax',
        kernel_regularizer=tf.keras.regularizers.l2(15)
        )])

model.compile(optimizer = tf.keras.optimizers.Adam(0.01),
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics='sparse_categorical_accuracy')


print('---------------load model-----------------')

# 加载训练好的模型的参数
model.load_weights('cnn.ckpt')
```

```
for data_set in data_set_total:
    result = [0,0,0,0,0,0,0,0,0,0,0,0]
    # 将 test 中 5 轮实验获得的为正的概率加和
    for x,y in data_set:
        x = np.array(x).reshape(1,x.shape[0],x.shape[1],1)
        y_ = model.predict(x)
        result[y-1] += y_[0][1]
    # 求出最大可能的行和列
    row = np.argmax(result[:6])
    col = np.argmax(result[6:])
    print(rc_dic[row][col])

# ----------------
# 第 3 问：自训练 cnn
# ----------------
import pandas as pd
import numpy as np
import os
import pickle
import sklearn
import tensorflow as tf
from                    tensorflow.keras.layers                    import
Dropout,Dense,Conv2D,Flatten,BatchNormalization,Activation
import matplotlib.pyplot as plt

seril_path = 'sri5.txt'

row_col = {'A':(1,7),'B':(1,8),'C':(1,9),'D':(1,10),'E':(1,11),'F':(1,12),
       'G':(2,7),'H':(2,8),'I':(2,9),'J':(2,10),'K':(2,11),'L':(2,12),
       'M':(3,7),'N':(3,8),'O':(3,9),'P':(3,10),'Q':(3,11),'R':(3,12),
       'S':(4,7),'T':(4,8),'U':(4,9),'V':(4,10),'W':(4,11),'X':(4,12),
       'Y':(5,7),'Z':(5,8),'1':(5,9),'2':(5,10),'3':(5,11),'4':(5,12),
       '5':(6,7),'6':(6,8),'7':(6,9),'8':(6,10),'9':(6,11),'0':(6,12)}
rc_dic = [['A','B','C','D','E','F'],
       ['G','H','I','J','K','L'],
       ['M','N','O','P','Q','R'],
       ['S','T','U','V','W','X'],
       ['Y','Z','1','2','3','4'],
       ['5','6','7','8','9','0']]

x_p = []
y_p = []
x_n = []
```

```
y_n = []


# 读入序列化后的数据
with open(seril_path,'rb') as f:
    x_p,y_p,x_n,y_n = pickle.load(f)


x_p = np.array([np.array(x)[:,[1,2,3,4,5,6,7,8,9,10,11,13,14,16,19]] for x in
x_p])
x_n = np.array([np.array(x)[:,[1,2,3,4,5,6,7,8,9,10,11,13,14,16,19]] for x in
x_n])
x_p = x_p.reshape(x_p.shape[0],x_p.shape[1],x_p.shape[2],1)
x_n = x_n.reshape(x_n.shape[0],x_n.shape[1],x_n.shape[2],1)
np.random.shuffle(x_p)
np.random.shuffle(x_n)

# 将训练数据分为 4：4：2，第一部分有标签，第二部分无标签，第三部分用来验证
a1 = 54
a2 = 108
b1 = 270
b2 = 540
x_p_train = np.repeat(x_p[0:a1],5,axis=0)
y_p_train = np.repeat(y_p[0:a1],5,axis=0)

x_p_mask = np.repeat(x_p[a1:a2],5,axis=0)
y_p_mask = np.repeat(y_p[a1:a2],5,axis=0)

x_p_test = np.repeat(x_p[a2:],5,axis=0)
y_p_test = np.repeat(y_p[a2:],5,axis=0)

x_train = np.vstack([x_p_train,x_n[0:b1]])
y_train = np.hstack([y_p_train,y_n[0:b1]])
x_mask = np.vstack([x_p_mask,x_n[b1:b2]])
y_mask = np.hstack([y_p_mask,y_n[b1:b2]])
x_test = np.vstack([x_p_test,x_n[b2:]])
y_test = np.hstack([y_p_test,y_n[b2:]])

np.random.seed(114)
np.random.shuffle(x_train)
np.random.seed(114)
np.random.shuffle(y_train)

# 在选择出来的 15 个通道上建立 cnn 模型，与上文基本一致
```

```
model = tf.keras.Sequential([
    Conv2D(filters=10, kernel_size=(1,15)
        ),
    Dropout(0.2),
    Conv2D(filters=50, kernel_size=(20,1),strides=10),
    BatchNormalization(),
    Activation('relu'),
    Dropout(0.2),
    Flatten(),
    Dense(100,activation = 'relu',
        kernel_regularizer=tf.keras.regularizers.l2(0.003)
        ),
    Dense(2,activation='softmax',
        kernel_regularizer=tf.keras.regularizers.l2(20)
        )])
model.compile(optimizer = tf.keras.optimizers.Adam(0.01),
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics='sparse_categorical_accuracy')
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath='cnn.ckpt',
    save_weights_only=True,
    save_best_only=True)

# 用有标签的样本训练初始模型
history = model.fit(x_train,y_train,batch_size=20,epochs = 50,
    validation_data=(x_test,y_test),validation_freq=1,callbacks=[cp_callback])

acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']
mask_x_colle = []
mask_y_colle = []
temp = []


# 在无标签数据集上选取预测概率高的数据，放入训练集，重新训练模型，
# 而后再从无标签样本中选择预测概率高的数据加入训练集，如此反复，
# 直到无标签数据集被取完或者无标签数据集中不存在预测概率高的数据
while(len(x_mask)>0):
    for x in x_mask:
        y_ = model.predict(x.reshape(1,x.shape[0],x.shape[1],1))
        prob = np.max(y_)
        if prob>0.8:
            mask_x_colle.append(x)
            mask_y_colle.append(tf.argmax(y_,axis=1).numpy().sum())
```

```
        else:
            temp.append(x)
    if len(mask_x_colle) == 0:
        break
    x_train = np.vstack([x_train,mask_x_colle])
    y_train = np.hstack([y_train,mask_y_colle])
    print('x_train.shape',x_train.shape,'y_train.shape',y_train.shape)

    history = model.fit(x_train,y_train,batch_size=20,epochs = 10,

    validation_data=(x_test,y_test),validation_freq=1,callbacks=[cp_callback])

    acc.extend(history.history['sparse_categorical_accuracy'])
    val_acc.extend(history.history['val_sparse_categorical_accuracy'])
    mask_x_colle = []
    mask_y_colle = []
    x_mask = temp
    temp = []

# 画出模型在训练集合和验证集合上的准确率变化图
plt.subplot(2,1,1)
plt.title('train_acc_and_test_acc')
plt.plot(acc,label='train')
# plt.xlabel('epoches')
plt.ylabel('acc')
plt.legend()
plt.subplot(2,1,2)
plt.plot(val_acc,label='test')
plt.xlabel('epoches')
plt.ylabel('acc')
plt.legend()
plt.show()

# -------------
# 第四问：svm
# -------------
import pandas as pd
import numpy as np
import os
import pickle
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.externals import joblib

x_collection = [[],[],[],[],[]]

# 读取数据集
io = pd.io.excel.ExcelFile(r'附件 2-睡眠脑电数据.xlsx')
dfs = pd.read_excel(io,sheet_name=None)
for key in dfs.keys():
    df = dfs[key]
    for i in range(df.shape[0]):
        index = df.iloc[i,0] -2
        sum = df.iloc[i,1]+df.iloc[i,2]+df.iloc[i,3]+df.iloc[i,4]
        data = [df.iloc[i,1],df.iloc[i,2],df.iloc[i,3],
            df.iloc[i,4],(100-sum)]
        x_collection[index].append(data)

for x in x_collection:
    np.random.shuffle(x)
total_num = len(x_collection[2])
# rate 为训练集占所有数据的比例，在保证一定准确率的基础上，本文希望寻找尽可能小的训练集
rate = 0.2
# num 为训练集的数量
num = int(rate*total_num)
print(num)
x_train = []
y_train = []
x_test = []
y_test = []
i = 0
for data in x_collection:
    x_train.extend(data[:num])
    y_train.extend([i for j in data[:num]])
    x_test.extend(data[num:])
    y_test.extend([i for j in data[num:]])
    i += 1
x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np.array(x_test)
y_test = np.array(y_test)
print(x_train.shape,y_train.shape,x_test.shape,y_test.shape)
```

```
np.random.seed(116)
np.random.shuffle(x_train)
np.random.seed(116)
np.random.shuffle(y_train)

# 以下为随机森林和 svm 的模型，这里将两个模型写在了一起，实际只运行其中一个
# 随机森林参数，树的数量 400，树的深度 60，在测试集分类准确率 71.84%
model = RandomForestClassifier(n_estimators=400,criterion='entropy',max_depth=60)
# svm 参数，惩罚系数 c=1，核函数为高斯核函数，核函数参数 gamma=2，在测试集分类准确率 71.35%
model = SVC(C = 1, kernel='rbf', gamma=2)
model = model.fit(x_train,y_train)
c = joblib.dump(model,"temp.m")
y_test_hat = model.predict(x_test)
y_test = y_test.reshape(-1)
print(y_test_hat)
print(y_test)
result = (y_test == y_test_hat)
acc = np.mean(result)
print("精准度：%.2f%%" % (100 * acc))

# ------------
# 第四位：nn
# ------------
import pandas as pd
import numpy as np
import os
import pickle
import sklearn
import tensorflow as tf
from tensorflow.keras.layers import Dropout,Dense,Conv2D,Flatten,BatchNormalization,Activation
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler


x_collection = [[],[],[],[],[]]

# 读取数据集
io = pd.io.excel.ExcelFile(r'附件 2-睡眠脑电数据.xlsx')
dfs = pd.read_excel(io,sheet_name=None)
for key in dfs.keys():
    df = dfs[key]
```

```
    for i in range(df.shape[0]):
        index = df.iloc[i,0] -2
        sum = df.iloc[i,1]+df.iloc[i,2]+df.iloc[i,3]+df.iloc[i,4]
        data = [df.iloc[i,1],df.iloc[i,2],df.iloc[i,3],
            df.iloc[i,4],(100-sum)]
        x_collection[index].append(data)

for x in x_collection:
    np.random.shuffle(x)

total_num = len(x_collection[2])
rate = 0.2
num = int(rate*total_num)
print(num)
x_train = []
y_train = []
x_test = []
y_test = []
i = 0
for data in x_collection:
    x_train.extend(data[:num])
    y_train.extend([i for j in data[:num]])
    x_test.extend(data[num:])
    y_test.extend([i for j in data[num:]])
    i += 1
x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np.array(x_test)
y_test = np.array(y_test)
print(x_train.shape,y_train.shape,x_test.shape,y_test.shape)

np.random.seed(116)
np.random.shuffle(x_train)
np.random.seed(116)
np.random.shuffle(y_train)

# 定义网络结构，两层全连接
model = tf.keras.Sequential([
    Dense(30,activation = 'relu'),
    Dense(5,activation='softmax',
        kernel_regularizer=tf.keras.regularizers.l2()
        )])

model.compile(optimizer = tf.keras.optimizers.Adam(0.01),
```

```
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics='sparse_categorical_accuracy')

cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath='nn.ckpt',
    save_weights_only=True,
    save_best_only=True)
history = model.fit(x_train,y_train,batch_size=7,epochs = 20,
    validation_data=(x_test,y_test),validation_freq=1,callbacks=[cp_callback])
model.summary()
acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']

plt.subplot(2,1,1)
plt.title('train_acc_and_test_acc')
plt.plot(acc,label='train')
plt.ylabel('acc')
plt.legend()
plt.subplot(2,1,2)
plt.plot(val_acc,label='test')
plt.xlabel('epoches')
plt.ylabel('acc')
plt.legend()
plt.show()
```